Planning and reconfigurable control of a fleet of unmanned vehicles for taxi operations in airport environment

ScuDo

Scuola di Dottorato ~ Doctoral School

WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation

Doctoral Program in Aerospace Engineering (29^{th}cycle)

Planning and reconfigurable control of a fleet of unmanned vehicles for taxi operations in airport environment

By

Giuseppe Sirigu

******

Supervisor(s):

Prof. Manuela Battipede, Supervisor

Prof. Piero Gili, Co-Supervisor

Prof. John-Paul Clarke, Co-Supervisor

Doctoral Examination Committee:

Prof. Eric Feron, Referee, Georgia Institute of Technology

Prof. Hamsa Balakrishnan, Massachusetts Institute of Technology

Prof. Giuseppe Del Core, Università degli Studi di Napoli "Parthenope"

Prof. Lorenzo Casalino, Politecnico di Torino

Prof. Giovanni Squillero, Politecnico di Torino

Politecnico di Torino

2017

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Giuseppe Sirigu
2017

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

To my Mom, you will always be with me

A mia Madre, sarai sempre con me

Acknowledgements

This dissertation conveys the work I have done during my Ph.D. since 2014 at the Politecnico di Torino and Georgia Institute of Technology (USA). During this period, I have worked with several researchers, each of whom has contributed to the realization of the work here presented. First, I would like to tank Prof. Manuela Battipede and Prof. Piero Gili for giving me the chance of doing the Ph.D. under their supervision, and for the continuous and invaluable inspiration support and guidance.

It has been a honor working with Prof. John-Paul Clarke as mentor at the Georgia Institute of Technology. He has introduced me into the area of operations research providing me with guidance support and inestimable advises. In the past two years, he has been a model on how to do research and he made grow my will of pursuing the research career. Hopefully, this has been only the start of a proficient collaboration.

Special thanks go to Prof. Dries Visser of TU Delf for his great and extremely useful suggestions to improve the final version of this dissertation.

Three years of intense research activity gave me the opportunity of meeting and collaborating with several Ph.D. and graduate students. My special thanks go to each one of them (apologizing if somebody has been forgotten), for being friends other than officemates: Mario Cassaro, Andrea Ferrero, Francesco Danzi, Harold Nikoue, Jose Magalhaes, Constantijn Sa, Raphaël Cohen, Heiko Udluft, and Tian Fu.

Last but not least, I would like to tank my dad for his support and for being a model and an inspiration on how to overcome the unfortunate events that life put us through.

I would finally thank all my friends: this great achievement would not have been possible without your support. Thank you!

Abstract

The optimization of airport operations has gained increasing interest by the aeronautical community, due to the substantial growth in the number of airport movements (landings and take-offs) experienced in the past decades all over the world. Forecasts have confirmed this trend also for the next decades. The result of the expansion of air traffic is an increasing congestion of airports, especially in taxiways and runways, leading to additional amount of fuel burnt by airplanes during taxi operations, causing additional pollution and costs for airlines. In order to reduce the impact of taxi operations, different solutions have been proposed in literature; the solution which this dissertation refers to uses autonomous electric vehicles to tow airplanes between parking lots and runways. Although several analyses have been proposed in literature, showing the feasibility and the effectiveness of this approach in reducing the environmental impact, at the beginning of the doctoral activity no solutions were proposed, on how to manage the fleet of unmanned vehicles inside the airport environment. Therefore, the research activity has focused on the development of algorithms able to provide pushback tractor (also referred as tugs) autopilots with conflict-free schedules. The main objective of the optimization algorithms is to minimize the tug energy consumption, while performing just-in-time runway operations: departing airplanes are delivered only when they can take-off and the taxi-in phase starts as soon as the aircraft clears the runway and connects to the tractor. Two models, one based on continuous time and one on discrete time evolution, were developed to simulate the taxi phases within the optimization scheme. A piecewise-linear model has also been proposed to evaluate the energy consumed by the tugs during the assigned missions. Furthermore, three optimization algorithms were developed: two hybrid versions of the particle swarm optimization and a tree search heuristic. The following functional requirements for the management algorithm were defined: the optimization model must be easily adapted to different airports with different layout

(reconfigurability); the generated schedule must always be conflict-free; and the computational time required to process a time horizon of 1h must be less than 15min.

In order to improve its performance, the particle swarm optimization was hybridized with a hill-climb meta-heuristic; a second hybridization was performed by means of the random variable search, an algorithm of the family of the variable neighborhood search. The neighborhood size for the random variable search was considered varying with inverse proportionality to the distance between the actual considered solution and the optimal one found so far. Finally, a tree search heuristic was developed to find the runway sequence, among all the possible sequences of take-offs and landings for a given flight schedule, which can be realized with a series of taxi trajectories that require minimum energy consumption. Given the taxi schedule generated by the aforementioned optimization algorithms a tug dispatch algorithm, assigns a vehicle to each mission. The three optimization schemes and the two mathematical models were tested on several test cases among three airports: the Turin-Caselle airport, the Milan-Malpensa airport, and the Amsterdam airport Schiphol. The cost required to perform the generated schedules using the autonomous tugs was compared to the cost required to perform the taxi using the aircraft engines. The proposed approach resulted always more convenient than the classical one.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviation

AI	Artificial Intelligence
AIP	Aeronautical Information Publication
AMVRPTW	Asimmetric Multi-trip Multi-vehicle routing problem with time windows
APU	Auxiliary Power Unit
ASPM	Aviation System Performance Metrics
ATC	Air Traffic Control
BADA	Base of Aircraft Data
BTS	Bureau of Transportation Statistics
ENAV	Ente Nazionale di Assistenza al Volo
EVOS	Electric Vehicles Optima Scheduling algorithm
FAA	Federal Aviation Administration
HC	Hill Climbing algorithm
HC-HPSO	Hybrid Particle Swarm Optimization using the Hill Climbing heuristic
HPSO	Hybrid Particle Swarm Optimization
ICAO	International Civil Aviation Organization

MVRPTW	Multi-trip Multi-vehicle routing problem with time windows
NLG	Nose Landing Gear
PIC	Pilot in Control
PSO	Particle Swarm Optimization
RGT	Recoverable Greedy Technique
RNS	Random Neighborhood Search algorithm
RNS-HPSO	Hybrid Particle Swarm Optimization using the Random Neighborhood Search heuristic
RVNS	Reduced Variable Neighborhood Search
TAP	Trajectory assignment problem

Symbols

$\%S$	Success rate	$[\%]$
α	Percentage of Iter for the fizing condition	$[-]$
α_s	Taxiway slope	$[deg]$
χ	Set of trajectories (solution)	$[-]$
$\mathbf{X}$	Set of solutions	$[-]$
$\Delta buff$	Buffer time step	$[s]$
ΔE^{bat}	Battery energy variation for a time step	$[kWh]$
ΔE_{tot}	Total variation of battery energy	$[kWh]$
Δt	Time step	$[s]$
ΔV	Speed step	$[m/s]$
Δ	Neighborhood half-size	$[-]$

Δ_{max}	Maximum value of Δ	$[-]$
Δ_{min}	Minimum value of Δ	$[-]$
δ_{sb}	Standby losses	$[-]$
η_c	Energy storage recovery efficiency	$[-]$
η_d	Energy storage discharge efficiency	$[-]$
η_{EM}	Electric motor efficiency	$[-]$
η_{in}	Power electronics discharge efficiency	$[-]$
η_{out}	Power electronics discharge efficiency	$[-]$
λ_o	Tractor center of gravity longitude	$[rad]$
ψ	Tractor heading angle	$[rad]$
ρ	Air density	$[kg/m^3]$
σ_U	Standard deviation of the tractor fleet utilization	$[-]$
θ	Threat deetection area inclination with respect to the East direction	$[rad]$
ε	Minimum clearance between a parked aircraft and any object	$[m]$
φ_o	Tractor center of gravity latitude	$[rad]$
A	Tractor drag area	$[m^2]$
a	Acceleration/Deceleration	$[m/s^2]$
b	Threat detection area short edge	$[m]$
buff	Set of buffer times	$[s]$
buff	Buffer time	$[s]$
c	Threat detection area long edge	$[m]$
c_1	Particle acceleration coefficient	$[-]$

c_2	Particle acceleration coefficient	$[-]$
C_D	Airplane drag coefficient	$[-]$
C_E	Energy consumption term of the cost function	$[kWh]$
C_T	Buffer time term of the cost function	$[s]$
C_X	Tractor drag coefficient	$[-]$
$combinations$	Number of possible combinations for the trajectory assignment problem	$[-]$
$Conn_{time}$	Time required to connect the tug to the aircraft	$[s]$
$Count$	Number of iterations without *gbest* improvements	$[-]$
d	Traveling distance	$[m]$
d_{min}	Shortest path distance between to points	$[m]$
d_{s-r}	Distance between stand and runway	$[m]$
$Disc_{time}$	Time required to disconnect the tug from the aircraft	$[s]$
DoD	Battery depth of discharge	$[-]$
E_{nom}^{bat}	Nominal battery capacity	$[kWh]$
e_{rec}	Fraction of braking recovery energy	$[-]$
$end_{slot_{asgd}}$	End time of the assigned runway slot	$[s]$
eq_{el}	number of particle elements equal tothe corresponding *gbest* elements	$[-]$
ERT	Earliest time at the runway for departure	$[s]$
f	Fitness value	$[-]$
F_a	Aerodynamic drag	$[N]$
F_i	Inertia force	$[N]$
F_r	Rolling friction resistance	$[N]$

f_r	Rolling resistance coefficient	$[-]$
F_s	slope resistance	$[N]$
g	Gravity acceleration	$[m/s^2]$
$\mathbf{G}$	Airport parking lot set	$[-]$
$gbest$	Best solution found so far by the whole swarm	$[-]$
$Iter$	Algorithm iteration	$[-]$
$job_{i,m}$	binary value indicating if the i-th tractor is assigned to the m-th mission	$[-]$
K	Particle swarm size	$[-]$
k_t	Buffer time term weighting coefficient	$[-]$
$l_{airplane}$	airplane length	$[m]$
$l_{tractor}$	Tractor length	$[m]$
lb	Variable lower bound	$[-]$
LRT	Latest time at the runway for departure	$[s]$
$MaxBuffer$	Maximum buffer time value	$[s]$
m_t	Tractor mass	$[kg]$
$m_{A/C}$	Aircraft mass	$[kg]$
max_{v_j}	Maximum velocity value of the j-th particle element	$[-]$
$maxCount$	Maximum number of iterations without solution improvements	$[-]$
$maxDoD$	Maximum battery depth of discharge	$[-]$
$maxIter$	Maximum number of iterations	$[-]$
$MaxIter_{RVNS}$	Maximum number of iterations for the RVNS algorithm	$[-]$
MGO	Maximum gate occupancy	$[s]$

MLW	Maximum landing weight	$[kg]$
MSG	Minimum separation at the gate	$[s]$
$MTOW$	Maximum take-off weight	$[kg]$
n	Particle length	$[-]$
$\mathcal{N}$	Solution neighborhhod	$[-]$
N_r	Number of runs	$[-]$
n_s	Tree search heuristic solution array length	$[-]$
Nb	Size of the set of buffer times	$[-]$
Nd	Number of tractor fleets in a specific airport	$[-]$
NF	Total number of departures/arrivals	$[-]$
Np	Size of the set of paths	$[-]$
NT	Tractor fleet size	$[-]$
$NTOs$	number of departing aircraft	$[-]$
NV	Size of the set of taxiing speeds	$[-]$
P	Mechanical requried power	$[kW]$
p	Path	$[-]$
$\mathbf{P}$	Set of paths	$[-]$
$P()$	Probability	$[-]$
$P_{aux_{idle}}$	Power consumed by auxiliary systems in idle	$[kW]$
P_{aux}	Power consumed by auxiliary systems	$[kW]$
PB_{buff}	Pushback buffer time	$[s]$
$pbest$	Best solution found so far by the single particle	$[-]$
$penalty$	Solution penalty value	$[-]$

$periods$	Time steps requried to travel a segment at a specific speed	$[-]$
$\mathbb{R}$	Set of real numbers	$[-]$
$\mathbb{R}^+$	Set of positive real numbers	$[-]$
$\mathbb{R}_0^+$	Set of positive real numbers excluding the 0	$[-]$
R_E	Earth radius	$[m]$
rel_{thold}	Iterations without *gbest* improvements after which fixed elements are released	$[-]$
S	Airplane wing surface	$[m^2]$
s	Tree search heuristic solution array	$[-]$
$safety\_gap$	Minimum time separation for safety	$[s]$
sep_{min}	Minimum runway separation	$[min]$
$separation$	Temporal separation between two aircrafts at a node	$[s]$
sf	Safety factor	$[-]$
SoC	Battery state of charge	$[-]$
$start_{slot_{asgd}}$	Start time of the assigned runway slot	$[s]$
t	Time	$[s]$
TOs	Set of departing aircraft	$[-]$
T	Time set	$[s]$
t_h	Planning time horizon	$[s]$
t_A	Arrival time at the parking lot	$[s]$
t_{oper}	Tractor operating time	$[s]$
t_{PB}	Pushback time	$[s]$
t_{RWYa}	Time at which the actual considered airplane starts occupying the runway	$[s]$

t_{RWYp}	Time at which the preceeding airplane starts occupying the runway	$[s]$
t_{TD}	Touch-down time	$[s]$
t_{TO}	Take-off duration	$[s]$
TAT	Turnaround time	$[s]$
TDA	Threat detection area	$[-]$
$Threshold$	Iteration after which the fixing algorithm is activated	$[-]$
TtC	Time to charge	$[s]$
U	Tractor utilization	$[-]$
$\bar{U}$	Average tractor fleet utilization	$[-]$
$\mathscr{U}$	Uniform distribution	$[-]$
ub	Variable upper bound	$[-]$
V	Taxiing speed	$[m/s]$
$\mathbf{V}$	Set of taxiing speeds	$[m/s]$
V_f	Taxi speed of the follower	$[m/s]$
$V_{E_{min}}$	Speed that corresponds to the minimum energy consumption	$[m/s]$
V_{high}	Lowest taxi speed to reach an assigned runway slot	$[m/s]$
$v_{k,j}$	Velocity of the j-th element of the k-th particle	$[-]$
V_{low}	Lowest taxi speed to reach an assigned runway slot	$[m/s]$
V_{max}	Maximum taxi speed	$[m/s]$
V_{min}	Minimum taxi speed	$[m/s]$
W	Particle inertia parameter	$[-]$
$w_{airplane}$	Airplane wingspan	$[m]$

$w_{tractor}$	Tractor width	$[m]$
x	Particle	$[-]$
x_{new}	Particle modified by the hybridization algorithm	$[-]$
$\mathbb{Z}$	Set of integer numbers	$[-]$
$\mathbb{Z}^+$	Set of positive integer numbers	$[-]$
$\mathbb{Z}_0^+$	Set of positive integer numbers excluding the 0	$[-]$

Chapter 1

Introduction

1.1 Motivation

1.1.1 Air traffic overview

Civil aviation has grown significantly in terms of both passengers (Fig. 1.1) and movements (Fig. 1.2) over the past decade [1]. Traffic forecasts suggest that this trend will continue in the mid- to long-term, resulting in a doubling of the number of passengers flying around the world to approximately 14 billion/year by 2029, growth that will mainly originate from emerging markets, and an increase of 2.5% in aircraft movements over the period 2015-2040 [1]. Data from the US Bureau of Transportation Statistics (BTS) highlight an increasing trend for the number of both domestic and international passengers (Fig. 1.3), whereas a contraction in the number of flights has been registered (Fig. 1.4) [2]. Both the number of passengers and the number of flights in the US has dropped between 2007 and 2009 by 8.1% and 10.23% respectively, probably as a consequence of the financial crisis. After this period, the number of passengers showed a positive trend, while the volume of movements decreased, especially in the domestic market. In Europe, analogous trends were recorded. Indeed, from 2008 to 2015, the number of passengers flying within and to/from Europe has substantially increased (Fig. 1.5), even though the total number of movements has remained almost constant (Fig. 1.6) [3]. As far as the historical data are concerned, the major contribution to the growth in passengers can be ascribed to passengers flying from/to outside the EU or flying internationally

intra-EU. Intra-EU flights represent the predominant market, with more than double the number of flights registered for the other market segments; as far as passenger numbers are concerned, the extra-EU and the intra-EU markets almost equal each other.

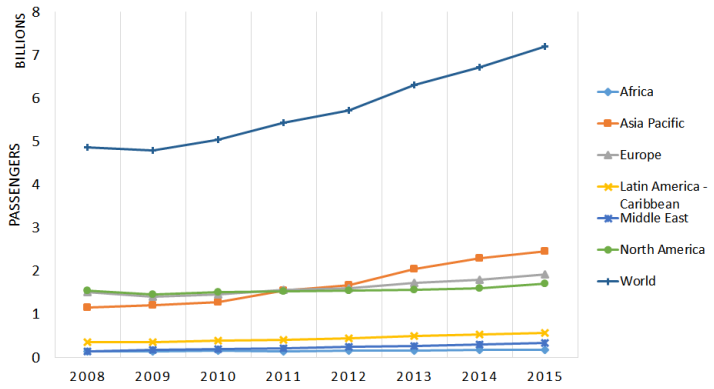


Fig. 1.1 Airport passenger trend per geographical area. Data source: Airport Council International.

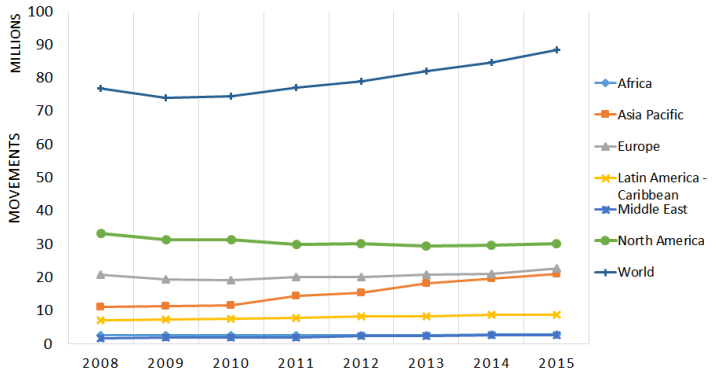


Fig. 1.2 Airport movements trend per geographical area. Data source: Airport Council International.

Fig. 1.3 Airport passenger trend per market segment and total in the US from 2002 to 2015. Data source: Bureau of Transportation Statistics T-100 Market data.

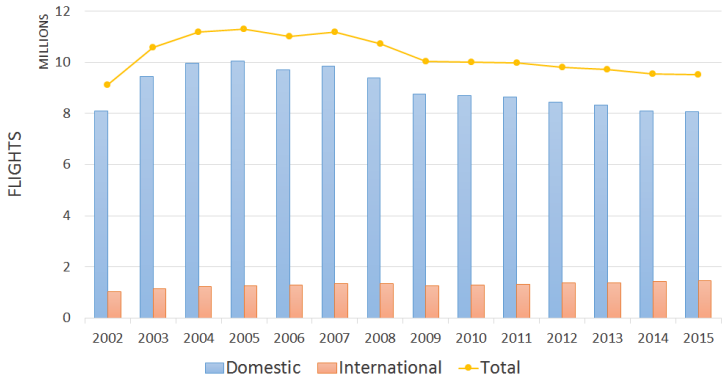


Fig. 1.4 Airport flights trend per market segment and total in the US from 2002 to 2015. Data source: Bureau of Transportation Statistics T-100 Segment data.

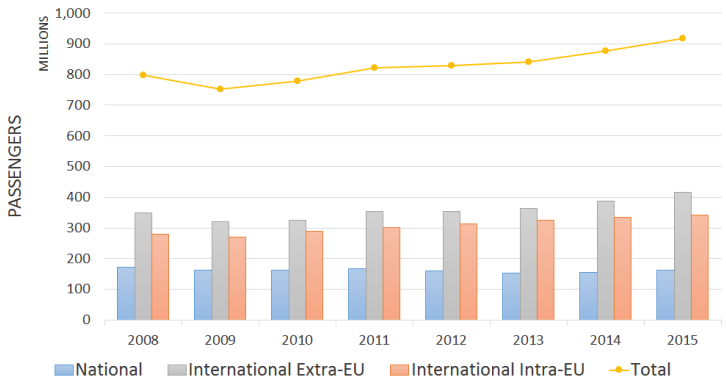


Fig. 1.5 Airport passenger trend per market segment and total in Europe from 2008 to 2015. Data source: Eurostat.

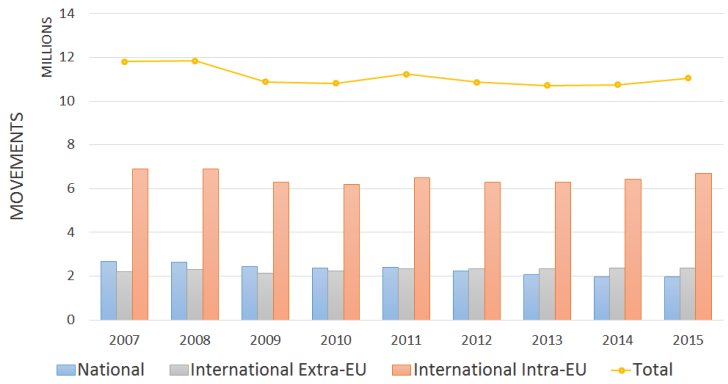


Fig. 1.6 Airport movements trend per market segment and total in Europe from 2008 to 2015. Data source: Eurostat.

Another metric that can be considered for the evaluation of air traffic is the fuel consumed by airplanes. The BTS provides the fuel consumption history for domestic and international flights (Fig. 1.7). As it can be noticed, despite the reduction in the number of flights in the US from 2009 to 2015, the fuel burnt has an almost constant trend with a 2% increase in 2015 with respect to 2009.

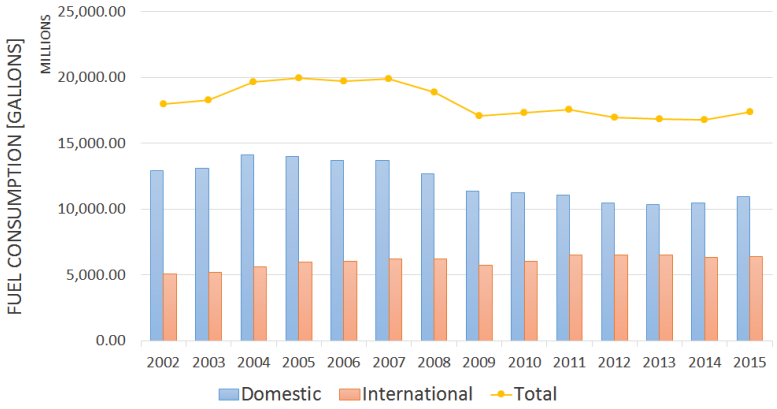


Fig. 1.7 Fuel consumption trend per market segment and total in the US from 2002 to 2015. Data source: Bureau of Transportation Statistics.

If history is indeed prologue, the main consequences of the forecasted air traffic increase will be increased airport congestion and longer queues at runways, leading to longer taxi times. Taxi time is defined as the time that an aircraft spends on the ground with engines on: either between pushback and take-off (taxi-out), or between touch-down and arrival at the gate (taxi-in). The Aviation System Performance Metrics

(ASPM) database, managed by the Federal Aviation Administration (FAA), provides historical data, including taxi time [4] data, for the evaluation of the performance of the US air transportation system. The average taxi-out and taxi-in time at the top airports in the US are shown in Figure 1.8 for the period 2002 to 2015. As may be seen, during the past decade the taxi time increased when the number of flights increased and decreased otherwise. However, starting from 2012, despite the negative trend in terms of the number of flights per year, the taxi time has increased, suggesting that there has been increasing congestion on the airport surface. The same trend has been registered for the taxi delay, namely the difference between the actual taxi time and the unimpeded one, estimated for each airport and airline as a weighted average for each year. The average delay among the airports is depicted in Fig. 1.9, for both the taxi-out and taxi-in [4]. The increased time spent by airplanes on the ground with the engines on causes additional fuel to be burnt, aggravating the environmental impact of the taxi phase and increasing the cost for airlines. A thorough investigation of the influence of congestion on the taxi-out time and, as a consequence, on the fuel burn and engine emissions is presented in [5]; analogous analyses could be made for the taxi-in time, as apron congestion also affects arrivals.

Fig. 1.8 Average taxi-out and taxi-in times for the ASPM airports from 2002 to 2015. Data source: Aviation System Performance Metrics.

The growth in air traffic also has an impact on safety. In the US, between 2008 and the end of 2014, 275 accidents and 25 incidents occurred during ground operations. The different types of accidents and their occurrence [6] are shown in Fig. 1.10a. Most of the accidents was caused by ground collisions (35.27%), followed by loss of control on ground (22.55%) and collision with terrain or objects (10.91%).

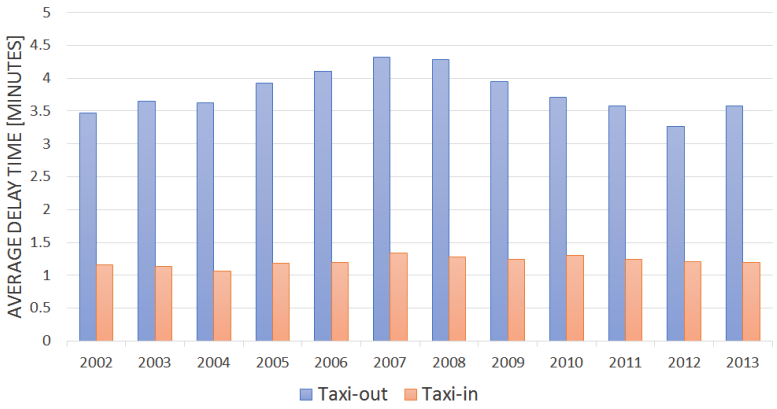
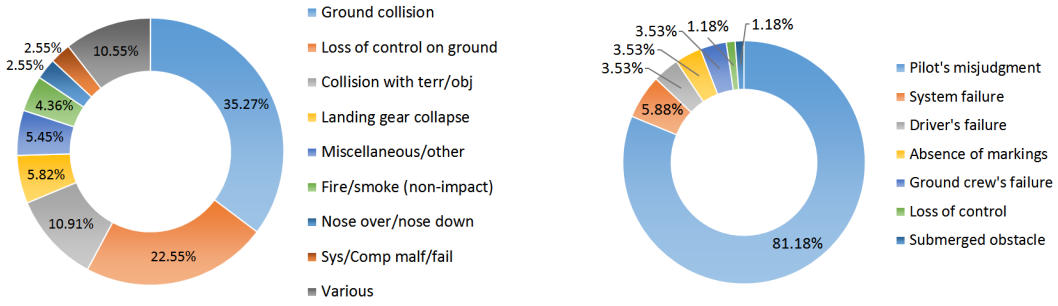


Fig. 1.9 Average delay with respect to the unimpeded taxi-out and taxi-in times for the ASPM airports from 2002 to 2013. Data source: Aviation System Performance Metrics.



(a) Taxi accidents types.

(b) Ground collision causes.

Fig. 1.10 Taxi accidents analysis in US airports from 2008 to 2014.

The sources of ground collisions are different: system failures could lead to an incorrect behavior of the aircraft, which can strike objects or other airplanes along the taxiway; the absence or malfunctioning of airport markings and visual aids, compromising the effectiveness of ground operations. Fig. 1.10b illustrates the distribution of the causes of ground collision accidents in the examined period; it follows that the main cause of accidents due to ground collisions was pilot's failures due to misjudgment of clearances (approximately 81%), whereas system failures and the absence of markings account for approximately 9% of the total ground collisions.

The failure of pilots to correctly assess the distance to other objects, static or dynamic, stem from poor pilot situational awareness, which could be generated by several causes: difficulties in maintaining an adequate visual lookout while taxiing; conventional landing gear (also known as tail-wheel landing gear, which consists

Fig. 1.11 Annual accidents occurred during ground operations in US airports from 2008 to 2014.

of two main wheels forward of the center of gravity and a small wheel or skid to support the tail), which prevents the pilot having a good view of the terrain and imposes "s" turns; procedures to be accomplished when taxiing; and instrumentation to be operated. Considering the number of accidents from 2008 to 2015, it may be seen that there was a substantial decrease during this period, as depicted in Fig. 1.11; thus, it can be deduced that technologies developed during those years to increase the safety during ground operations, together with enhanced pilot training, proved to be effective, even though further improvements are still required.

It appears that existing technologies are not sufficient to meet the objective, fixed by the main aeronautical authorities, of making airspace systems, including airports, safer and greener, despite the growth of the number of aircraft, moving every day all over the world. Thus, new procedures and equipments if ground operations are to be performed in a safer manner, in all conditions, with lower environmental impact. Optimization of ground operations is a cost effective way of achieving the aforementioned goals.

1.1.2 Ground operation optimization

The optimization of airport ground operations has been a popular topic among the researchers in the past decades. As described in [7], operations research techniques can be successfully applied to different branches of the air transportation industry, including airside operations. A tree search has been proposed in [8] to solve the run-

way occupancy problem and define the optimal departure sequence that maximizes runway throughput, reduce taxi time and delays, and manage controllers' workload. Simulation has also been used to study and characterize the dynamics of busy airport surfaces. Besides, accurate simulators can be used to test and evaluate departure control strategy to alleviate departure traffic congestion in busy airports [9]. Two queuing models and an integer programming model have been proposed in [10] to simulate respectively the taxi-in and taxi-out flows and the airline decision-making for tournaround processes. The scope of this simulation was that of predict with good accuracy airport congestion and test control strategies to increase airport efficiency.

Among the ground operations, the airplane taxi phase has received particular attention, in order to reduce its environmental and economical impact. Previous research in the area of ground operations mostly focused on the reduction of the time spent with the engines on, without taking directly into consideration the fuel consumption, as it had been assumed to be proportional to the taxi time. Gotteland and Durand [11–13], applied a genetic algorithm and a branch & bound based algorithm to the reduction of the total taxi time, increased by the time spent in lengthened trajectories. Taxi time and delay reduction is the goal of the mixed integer programming algorithm proposed by Guépet et al. [14] and of the genetic algorithm presented in [15]. A linear programming model is established in [16], which is used within a genetic algorithm to minimize the taxi time. The paper by Jingnan and Qing [17] presents two particle swarm optimization algorithms to minimize the total taxiing time of all the aircraft by ordering the arrival of the aircraft at each taxiway segment. The vertex-based label-setting Quickest Path Problem with Time Windows (QPPTW) algorithm, introduced in [18], aims to absorb as much delay time as possible at the stand, decreasing the engine-on time and, indirectly, the fuel consumption. A modified version of the QPPTW, presented by Stergianos et al. [19], takes into account also the time required to perform the pushback, leading to a more realistic model. The improved $A*$ algorithm proposed in [20] implements a conflict avoidance multi-objective optimization function to include the conflict information inside the path optimization.

In recent years, the environmental impact of ground movements has gained more attention, leading to the development of optimization models that explicitly consider the fuel consumption during the optimization process, with the final goal of having a better estimate of the fuel burn relative to the simply time proportionality model. Weiszer et al. [21] proposed a heuristic approach aiming at the optimization of the

speed profile for the family of k-best taxi routes, generated using the k-QPPTW algorithm; this approach has been improved for real-time applications using a pre-computed database of optimized speed profiles, for taxi route building blocks [22]. A multi-objective speed profile optimization algorithm for a trade-off between taxi time and fuel consumption is presented in [23]; two different models are proposed for the fuel consumption evaluation: one based on the international civil aviation organization (ICAO) emission database, and one that exploits the Eurocontrol base of aircraft data (BADA). A two-stage speed profile design that uses a particle swarm optimization is presented in [24] and in [25]: the first stage of the algorithm allocates speed values at defined control points along the path, in order to ensure a smooth profile; in the second stage, the performed speed allocation is used as input for an algorithm that optimizes the speed profile for each taxi segment. Kjenstad et al. [26] have proposed and validated a heuristic approach to decompose the arrival, surface, and departure management problems, in order to solve the routing, sequencing, and deconfliction problems in subsequent phases.

Alternative solutions to perform the taxi phase, based on maintaining the aircraft main engines off, were proposed in past years in both academia and industry. Safran, in collaboration with Honeywell, and WheelTug separately developed similar solutions that use electric motors mounted on the nose landing gear (NLG) of the airplane to motion it on the airport surface [27, 28]; in both cases, the electric motors are powered by the auxiliary power unit (APU). The main benefits of this solution have been quantified in a lower amount of fuel burnt by the main engines, reduced tug operations and decreased foreign object debris damage. Besides, the pushback capability granted by the electric motors enables to save time, as no tractors are required to move from the stand. On the other hand, these systems cause an increase in the fuel that is burnt by the APU and additional fuel consumption during the flight, due to the supplementary weight of the taxiing system. Furthermore, the taxi speed granted by this devices is lower than the airplane taxi speed when powered by its own engines, and the energy absorbed by the electric motor can prevent the engine start-up from the APU until the runway is reached; these features can generate delays, especially in airports where the taxiway width does not permit overtakes near the holding positions [29]. When performing taxi operations by means of alternative solutions, it must be taken into account the necessity for aircraft engines to warm up prior departure for a period that ranges from 2 to 5 minutes, depending on the engine model and the external temperature [30].

The use of electric tugs to tow aircraft between stands and runways has been proposed and analyzed in the literature for semi-autonomous [31] and fully autonomous [32–35] tractors. The Taxibot system provides the pilot with steering and braking capabilities through an innovative nose landing gear interface, thereby the pilot can act as if he were taxiing in a conventional way. The system grants a taxi speed up to 23 *knots*, same as actual airplane taxi speed. During each phase of the taxi, a pilot in control (PIC) is ensured, meeting all the safety, accountability and regulatory requirements [31].

A futuristic approach that uses fully autonomous taxi systems can overcome some of the weaknesses of the semi-autonomous solution: the taxiing speed can be better controlled by the onboard computer than by the pilot, leading to a better accuracy in tractor positioning in time, a reduced workload required for pilots, with the possibility of head-down operations during taxi. Furthermore, tractors can be equipped with sensing systems able to enhance their situational awareness, thereby automatically reacting if potential hazards are detected. This capability can reduce the occurrence of incidents and accidents.

Although several papers analyzing the feasibility and effectiveness of fully autonomous taxiing systems have been proposed in literature, little work has been done in terms of articulating how the fleet management should best be performed in an airport environment, thereby justifying the need to research systems that are able to provide conflict-free schedules for the tractor autopilots, via the solution of the associated planning and scheduling problems.

1.2 Overview of planning and scheduling algorithms

In automated planning, researchers study the process of choosing and organizing actions by anticipating their outcomes, thereby achieving, with certain accuracy, a predefined objective [36, 37]. Scheduling addresses the problem of how to perform a set of actions given limited resources in a limited amount of time [38, 36, 39]. Resources are defined as entities that can be used in order to perform an action; resources can be:

- reusable: resources that return available to other actions after their exploitation;

- consumable: resources that deteriorate when used but can be restored by other actions.

Addressing the two problems separately might not work when complex systems are considered; therefore, an integrated approach that takes into account both planning and scheduling is required [36]. Polynomial algorithms have been implemented to solve scheduling problems that have been proved to be polynomially solvable, such as the *single-machine*, the *flow-shop* and the *open-shop* problems. Exact approaches based on branch and bound have been widely used [40–44]. Linear programming techniques [45, 46] have also been implemented to solve this family of problems.

Modern applications, in which the complexity of systems to be optimized is ever increasing, led to the definition of problems for which exact algorithms are not tractable, as they would require excessive time to reach solutions. Therefore, approximation techniques, such as approximate dynamic programming (e.g. Q-learning) [47–49] and greedy algorithms [50, 51] have been used to reach solution in a reasonable time. Heuristic and meta-heuristic algorithms have also been successfully applied to complex scheduling and combinatorial optimization problems. Heuristics have been proposed in literature to solve scheduling and routing problems [52–54].

Meta-heuristics can be divided in local and global search algorithms. The first ones, such as the tabu search [55–57] and the hill-climbing (HC) [58, 59], iteratively move from a tentative solution to a neighbor solution in the search space; however, local search can get stuck in local optima if no improving solutions are present in the actual neighborhood. Global search algorithms were proposed to overcome this problem; an important branch of global search meta-heuristics is represented by population-based algorithms. In these approaches, the population components collaborate by sharing information to reach the optimization goal; several applications of population-based algorithms to scheduling and assignment problems have been proposed in literature for the ant colony algorithm (AC) [60, 61], the genetic algorithms (GA) [62–65] and for particle swarm optimization (PSO) [66–70].

For the sake of enhancing the performance of meta-heuristics, hybridization techniques can be effectively employed; hybrid algorithms combine the properties of global and local search to balance exploration and exploitation and to achieve better solutions [71]. Different algorithms have been used for the meta-heuristic

hybridization: other (meta-)heuristics, constraint programming, tree search methods, problem relaxation, and dynamic programming [72].

In the present work, hybrid versions of the particle swarm optimization and a novel form of greedy technique, named Recoverable Greedy Technique (RGT), were used to solve the autonomous taxi scheduling problem. In the following subsections, a general overview of these optimization techniques is presented.

1.2.1 Particle swarm optimization

Particle swarm optimization is a population-based meta-heuristics that was first proposed by Kennedy and Eberhart in 1995 [73]. They have been inspired by the social behavior of some animals, like bird flocking, which collaborate to achieve a common goal. In the same way, the particles composing the swarm cooperate to find the best solution in the problem search space. Each particle benefits from its experience (*pbest*) and from the swarm's one (*lbest* or *gbest*); *pbest* represents the best solution found by each particle so far. The swarm can be divided into neighborhoods of defined size (usually 15% of the entire swarm); in this case, each particle can move towards the best-neighbor previous position *lbest* or can be headed for the actual best neighbor [74]. A different approach considers the best solution *gbest* found so far by any particle in the whole swarm; the global solution can be seen as a special case of the local one, where the neighborhood size equals the swarm size. The PSO can be effectively applied to both continuous and discrete problems; its stability and convergence properties were largely discussed in [75] and [76]. The main strength of PSO-based algorithms is that, conversely to other meta-heuristics (i.e. genetic algorithms), the PSO has fewer parameters to be tuned and can be implemented in few lines of code using fundamental mathematical functions, thereby requiring less computational effort, without affecting the quality of the solution [77].

An initial family of K particles, each composed by n elements and associated velocities, are randomly generated in order to spread as much as possible the particles among the search space and, thus, to increase the probability of reaching the optimal solution. The *pbest* and *gbest* positions are determined by means of the fitness value f of the particles. The velocity $v_{k,j}$ of each element j of the particle x_k is randomly generated considering the *pbest* and *gbest* solutions, as described in Eq. 1.1.

$$v_{k,j} = W \cdot x_{k,j} + c_1 \cdot rand_1 \cdot (pbest_{k,j} - x_{k,j}) + c_2 \cdot rand_2 \cdot (gbest_j - x_{k,j}) \quad (1.1)$$

The inertia parameter W is used to control the balance between exploration and exploitation: a larger inertia value leads the particles to explore new areas of the search space and prevents the particles being trapped in local optima; lower inertia values push towards a refinement of the actual position. The c_1 and c_2 weights are called acceleration coefficients and determine whether the particle is guided more by a cognitive intelligence *pbest* or by a social one *gbest*. $rand_1$ and $rand_2$ are random numbers between 0 and 1. The velocity is bounded by the value max_v in order to prevent the particles from escaping from a possible optimal solution. Once the velocity has been computed, the particle position in the search space is updated following Eq. 1.2.

$$x_{k,j} = x_{k,j} + v_{k,j} \quad (1.2)$$

The process described above and the particle fitness evaluation are repeated until a stopping condition is met; the algorithm pseudocode is reported in Algorithm 1.

Algorithm 1 Particle swarm optimization pseudocode.

```
1: Generate K random particles
2: Initialize velocities v_{k,j} between 0 and 1
3: while (Stopping condition) = FALSE do
4:     Evaluate the fitness for each particle
5:     Determine the best solution pbest visited so far
6:     Determine the global best solution gbest visited so far
7:     Compute the particle velocities
8:     Update the particle position
```

1.2.2 Greedy algorithms

Greedy algorithms are optimization schemes characterized by the fact that, at each decision time, they make always the most convenient choice at that moment. Locally optimal choices not always lead to a globally optimal solution; however, for optimization problems characterized by an optimal substructure, greedy algorithms

have been proved to reach the global optimum [78]. Optimal substructure problems are defined as problems where the optimal solution contains the optimal solutions to its subproblems. Unlike dynamic programming algorithms, which work in a bottom-up fashion, greedy techniques usually work top-down; this means that they make choices only considering what is best at the actual decision point and then solve the remaining subproblems. The nature of the choice can depend on the previous ones, but never on the future choices or subproblems.

1.3 Contribution of the dissertation

The main focus of this dissertation is the development, implementation and verification of a tool to provide conflict-free schedules to a fleet of autonomous vehicles, for the airplane ground handling and the execution of just-in-time runway operations. Three algorithms for the solution of the trajectory assignment and tug dispatch problems were developed:

- a hybrid version of the particle swarm optimization that uses a hill-climbing metaheuristic as local search (HC-HPSO);
- a hybridization of the particle swarm optimization by means of an algorithm of the family of the variable neighborhood search (RNS-HPSO);
- a tree-search (TS) heuristic algorithm that employs the Recovery Greedy Technique to find the departure sequence that leads to taxi trajectories with minimum cost.

Along with the three optimization schemes, two mathematical formulations of the autonomous taxi problem, one based on discrete time and one on continuous time, and a fast-computation energy consumption model, to be implemented within the optimization algorithm, were developed. To the best of author's knowledge, none of the developed algorithms have ever been proposed, especially for the solution of the autonomous taxi problem. In fact, despite the growing literature on the topic of autonomous tugs, at the start of the author's research work, no paper was discovered that focused its attention on algorithms for the management of a fleet of autonomous tractors, dedicated to the airplane ground handling. The profitable implementation of this futuristic solution truly depends on the development of a tool that provides

reliable schedules to the tractor autopilots. In this sense, the present work represents a starting point for the development of more advanced algorithms to deal with this extremely complex problem.

1.4 Dissertation organization

The mathematical models used to describe the problem are presented in Chapter 2, along with a computational complexity analysis, and a thorough description of the tractor energy consumption model. The three optimization algorithms are analyzed in Chapter 3, where a parameter tuning analysis and a proof of convergence is also provided. In Chapter 4, the results of seven test cases are presented for three airports with different characteristics: the "Sandro Pertini" Turin-Caselle airport, the Milan-Malpensa airport, and the Amsterdam airport Schiphol. For each test case, multiple runs were performed using each algorithm with both mathematical models; results are compared in terms of cost function and required computational effort. Finally, the main remarks and final conclusions are provided in Chapter 5.

Chapter 2

Autonomous taxi system

The proposed autonomous fleet management system has been designed to solve three main problems: the departure sequence scheduling, the pushback scheduling, and the taxi route planning. Furthermore, the tug dispatch problem is solved, assigning an autonomous vehicle to each mission (departures and arrivals). The output of the algorithm is a schedule for each tug that enables just-in-time runway operations: the taxi-in starts as soon as the aircraft clears the runway and connects to the tug, whereas the airplane is delivered to the runway at the time when it should takeoff. The structure of the fleet management system is shown in Fig. 2.1 [79].

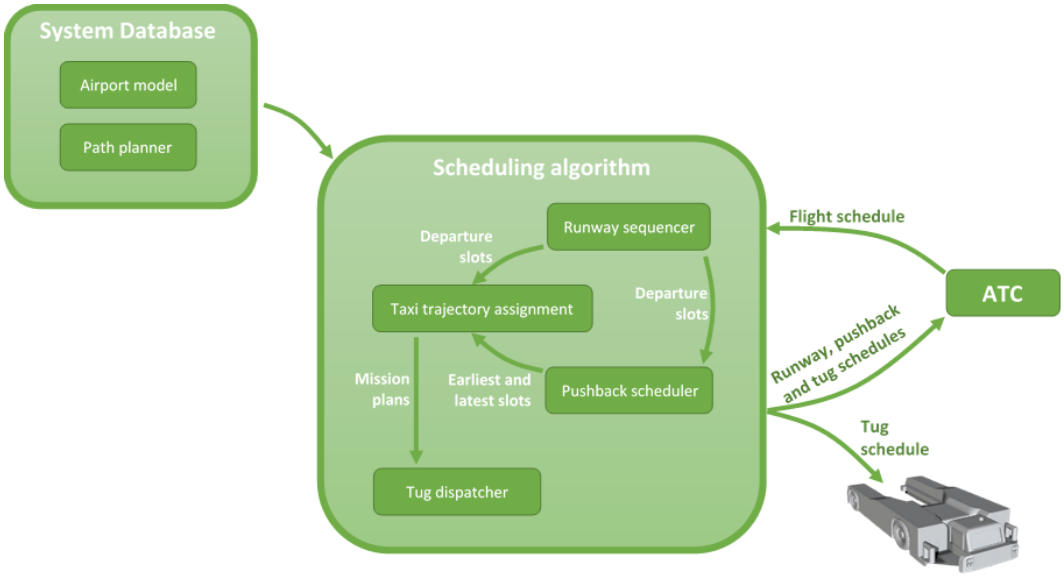


Fig. 2.1 Fleet management system.

The following functional specifications are defined for the routing and scheduling algorithm:

- Reconfigurability: the algorithm has to be easily adapted to modifications in the airport layout;
- Reliability: the schedule must always be free of conflicts among the agents;
- The computational time required to process a time horizon $t_h = 1\ h$ has to be less than 15 *min*.

The last functional specification was defined to guarantee that the algorithm always work with reliable arrival and departure times. In fact, in most of the cases, the arrival or departure times can be estimated with good accuracy within one hour horizon. Besides, the algorithm is created to be run 15 *min* before every hour to compute the schedule for the subsequent hour.

In order to define the tug schedules, the algorithm assigns a trajectory to each tractor mission based on the data provided by the runway and the pushback schedulers, and evaluates the cost associated with it in terms of the electric energy consumed. After the best set of non-conflicting trajectories has been found, the tasks are assigned to each tractor in the fleet, thereby maximizing the tug utilization. Finally, the generated schedules are communicated to air traffic control (ATC) and to the tugs.

If the considered airport is characterized by the presence of more than one runway, the airplanes might be required to cross one or more active runways to reach their assigned runway or stand. To prevent excessive numbers of crossings and consequently the risk of runway incursions, the proposed autonomous taxi system is designed to tow the aircraft to the closest runway to the apron or vise versa. Therefore, aircraft must use their own engines to go from the disconnection point to the take-off runway or from the landing runway to the connection point. As far as departing aircraft are concerned, the time spent traveling from the disconnection point to the take-off runway with their own engines enables the engine to warm up before departure. In order to define the time at which a landing airplane reaches the connection point, a constant taxi speed of 10 m/s is considered; if the runway to be crossed is busy, the airplane will stop at the holding point until the runway is clear.

A description of the data and the models used by the optimization algorithms at lie at the heart of the autonomous taxi system is presented in this chapter. It is

organized as follows: in Section 2.1, the tool that is introduced helps the user to create the airport surface model, which represents the input of the path generation algorithm described in Section 2.2. In order to evaluate the energy consumed to perform the assigned missions, an energy consumption model that takes into account the characteristics of the airplanes to be towed has been defined; this is outlined in Section 2.3. Finally, the mathematical problem formulation is presented for the trajectory assignment and the tractor dispatch problems in Section 2.4 and Section 2.5 respectively.

2.1 Airport model

The first operation to be performed is the definition of the spatial domain in terms of geographical coordinate system; the aeronautical authority can provide those information, as aircraft parking/docking, taxiways and runway locations, through the Aeronautical Information Publication (AIP). However, in order to be properly used by a path planning algorithm, these data must be discretized and collected in organized form. Aircraft stands, runway entries and taxiway intersections can be modeled as nodes of a directed weighted graph, where arcs represent taxiway segments and the associated weights their length. This representation is really powerful and can be used in conjunction with the most common path planning algorithms.

A graphical user interface (GUI) was developed in MATLAB® environment to make the preprocessing more user-friendly and to easily access different airport databases. The GUI, showing the discretization of the *Sandro Pertini* airport, is reported in Appendix A, together with an explanation of the main functions behind the GUI.

2.2 Path generation

From the tractor point of view, each taxi mission has three phases: a central towing phase (Phase 2), in which the tractor tows the airplane and two repositioning phases (Phase 1 and Phase 3), where it moves between the depot and the aircraft, as showed in Fig. 2.2. The portion of mission labeled PB_{buff} represents a delay up to $MaxBuffer = 10\ min$ on the pushback time, which is admitted in order to deliver the

aircraft at an available departure slot. At the end of the first and second phases of each mission, the tractor must respectively connect to $(Conn_{time})$ and disconnect from $(Disc_{time})$ the airplane; the connection period was considered with duration 2 *min* and the disconnection period was set to 1 *min*. The strategy of having the tractors always traveling back to the depot might not always be efficient. In fact, there might be some cases in which, after delivering an airplane to the runway or to the stand, it would be more efficient for the tractor to directly start a new mission. This is an important limitation of the proposed methodology.

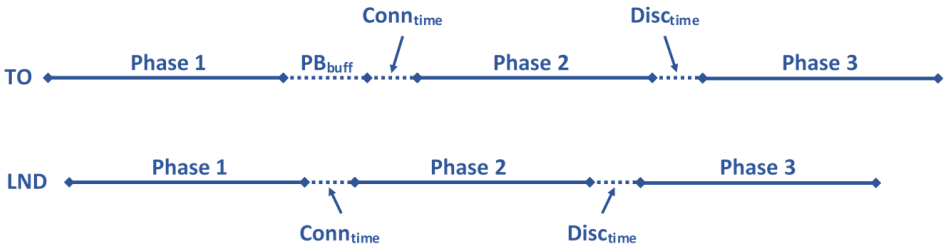


Fig. 2.2 Mission representation.

For each phase, the first problem to be addressed is the path definition. The shortest path between the origin and the destination would be the best solution, as it requires the shortest time to be traveled for a given speed. However, the shortest path might not always be feasible, because, for example, another agent might be traveling on the same path, or the destination has not yet be cleared from other airplanes. Therefore, an algorithm able to find more than one path between origin and destination has been implemented.

Several algorithms have been proposed in literature to select the shortest path between two nodes in a graph, such as the Dijkstra's [80, 81], the A* [82, 83] and the breadth-first search [84, 85] algorithms [86]. However, there is no widely used algorithm for finding all the possible paths between two nodes. A modified version of the breadth-first search (BFS) algorithm was implemented (Algorithm 2); the proposed approach does not take into account the distance between the nodes during the path construction and provides paths visiting each node no more than once. The matrix *graph* is the connectivity matrix of the graph representing the domain. The inputs x and d are respectively the indexes of the currently considered node and of the destination. The output of the algorithm contains all the paths going from a to b.

Algorithm 2 Modified breadth-first search algorithm.

Require: a,b

```
function BFS(graph, x, d, p)
    Append node x at the temporary path p
    Find all the neighbors of x
    for i in neighbors do
        if i = d then
            Save the temporary path p
        else if i ∉ p then
            BFS(graph, i, d, p)
    Delete the last element of p
```

The computational time required to find all the paths between two vertices in a large graph can compromise the efficiency of the fleet management system. Therefore, a database of the possible paths between all the main locations inside the airport surface is created and the trajectory assignment algorithm can access it during the optimization process.

However, as far as big airports are considered, the number of existing paths, given an origin and a destination, can be considerable (over 100 in the Schiphol airport), with most of them resulting too long to effectively improve the optimization solution. Consequently, an algorithm able to find a limited number of paths is required. Several algorithms for the solution of the k-shortest path problem have been proposed [87–89]. For the present application, Yen's algorithm has been implemented to find the $k = 5$ shortest paths, thereby creating a database for each considered airport, containing the 5 shortest paths for each starting/ending node pairs of the taxi grid [87].

2.3 Energy consumption model

The energy consumption of the electric tugs is modeled using the approaches proposed in [90–92]. The mechanical power P (Eq. 2.1), required to move the tractor at the speed V, is computed considering the forces acting on it (Eq. 2.2-2.5): the aerodynamic drag F_a, the rolling friction F_r, the inertial force F_i and the slope resistance F_s.

$$P = (F_a + F_r + F_i + F_s) \cdot V \tag{2.1}$$

Both the tractor itself and the towed airplane must be considered during the force computation.

$$F_a = \begin{cases} \frac{1}{2} \cdot \rho \cdot V^2 \cdot (A \cdot C_X + S \cdot C_D) & if\ towing \\ \\ \frac{1}{2} \cdot \rho \cdot V^2 \cdot (A \cdot C_X) & otherwise \end{cases} \tag{2.2}$$

The aerodynamic drag depends on the air density ρ, which is a function of the airport altitude, the taxiing speed V, the drag area of the tractor A, the airplane wing surface S and on the aerodynamic coefficients C_X and C_D of the tractor and airplane respectively. At the state of the art of the proposed application, the wind speed is not taken into account, but it will be taken into account in future developments. The other forces acting on the tractor are functions of the mass of the tractor m_t and of the airplane $m_{A/C}$, the gravity acceleration $g = 9.81\ m/s^2$, the acceleration/deceleration $a = \pm 1.2\ m/s^2$ during speed increase/decrease phases, set to keep a good level of comfort for the passengers $(0.12\ g)$, the rolling resistance coefficient $f_r = 0.015$, and the taxiway slope α_s.

$$F_r = \begin{cases} \left(m_t + m_{A/C}\right) \cdot g \cdot f_r \cdot \cos \alpha_s & if\ towing \\ \\ m_t \cdot g \cdot f_r \cdot \cos \alpha_s & otherwise \end{cases} \tag{2.3}$$

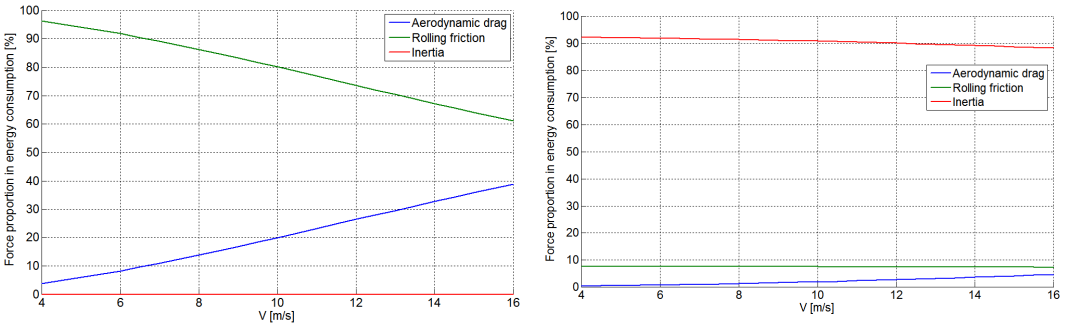
$$F_i = \begin{cases} \left(m_t + m_{A/C}\right) \cdot a & if\ towing \\ \\ m_t \cdot a & otherwise \end{cases} \tag{2.4}$$

$$F_s = \begin{cases} \left(m_t + m_{A/C}\right) \cdot g \cdot \sin \alpha_s & if\ towing \\ \\ m_t \cdot g \cdot \sin \alpha_s & otherwise \end{cases} \tag{2.5}$$

In the present study, the taxiway design standards are assumed to meet international regulations that require the taxiway slope being $\alpha_s \leq 3\%$ [93]. With these assumptions, the equation for the required mechanical power becomes:

$$P = (F_a + F_r + F_i) \cdot V \tag{2.6}$$

To evaluate the weight of each force in the required power, the above equations are applied to the case of a route of length $d = 1000\ m$. In Fig. 2.3, the weight of each force is reported in terms of percentage of the total required power for different speed values for both the constant speed (Fig. 2.3a) and the acceleration cases (Fig. 2.3b). As expected, the role of the aerodynamic drag grows with the speed, up to reach about 40% of the total required power in the constant speed case. When the acceleration is considered, the main component of the required power is the inertia, which counts for about 90% of the required power.



(a) Constant speed case

(b) Acceleration case

Fig. 2.3 Analysis of the weight of each force on the required power.

The energy variation at each time step is computed by means of Eq. 2.7, where η_d/η_c is the energy storage discharging/recovery efficiency, η_{out}/η_{in} is the discharging/recovery efficiency of the power electronics, η_{EM} the electric motor efficiency and P_{aux} considers the power consumed by auxiliary systems (e.g. lights, computers, etc.). The capability of braking energy recovery is defined by the percentage of energy that can actually be recovered e_{rec}.

$$\Delta E^{bat} = \begin{cases} \dfrac{\Delta t}{\eta_d \cdot \eta_{out} \cdot 3600} \cdot \left(\dfrac{P}{\eta_{EM}} + P_{aux} \right) & if\ P \geq 0 \\ \dfrac{\Delta t \cdot \eta_c \cdot \eta_{in} \cdot e_{rec}}{3600} \cdot (P \cdot \eta_{EM} + P_{aux}) & if\ P < 0 \end{cases} \tag{2.7}$$

The state of charge (SoC) of the battery pack can be updated considering also the standby losses δ_{sb} (Eq. 2.8). In order to prevent a rapid battery wear, they are not allowed to fully discharge; thus, the total battery capacity must be higher than the usable (E^{bat}_{nom}).

$$SoC_i = SoC_{(i-1)} - \delta_{sb} + \Delta SoC = SoC_{(i-1)} - \delta_{sb} - \frac{\Delta E^{bat}}{E^{bat}_{nom}} \tag{2.8}$$

The proposed model is deeply influenced by the choice of the time step Δt, as described in Fig. 2.4 for a route of length $d = 1000\ m$ and the tractor characteristics reported in Table 2.1; indeed, the accuracy of the solution increases as the time step decreases.

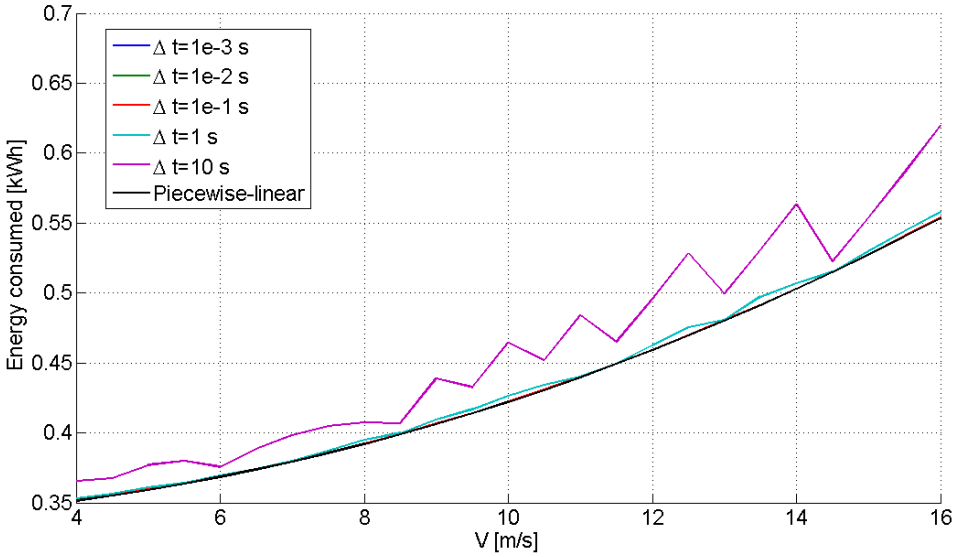


Fig. 2.4 Analysis of the time step influence on the consumed energy for different values of the taxiing speed. Constant speed was considered during the whole segment.

However, very small values of Δt would require excessive computational resources to manage the tractor schedule. Therefore, each phase of the missions is divided into three segments (Fig. 2.5): acceleration, constant speed, and deceleration

Table 2.1 Tractor mechanical and electrical specifications.

C_X	A $[m^2]$	m [kg]	η_{EM}	η_{out}	η_d	η_{in}	η_c	e_{rec}	δ_{sb}	P_{aux} [kW]	E^{bat}_{nom} [kWh]
0.6	6.90	10400	90%	98%	95%	95%	95%	90%	0	1.5	60

segments. A database of the energy consumption and the time and space required to accelerate and decelerate is precomputed for the acceleration from standstill and deceleration back to standstill, using the consumption model presented above and $\Delta t = 10^{-3}$ s. The database considers the tractor specifications, whether it is towing an airplane or not and the characteristics of the towed aircraft, e.g. drag coefficient and weight range (maximum take-off weight (MTOW) for departures and maximum landing weight (MLW) for arrivals).

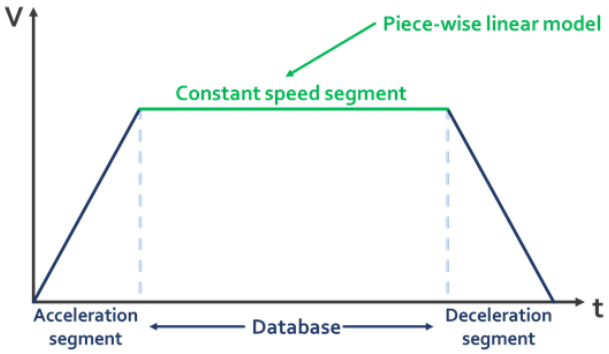


Fig. 2.5 Speed profile for each path segment.

As far as the constant speed segments are considered, a piecewise-linear model for the energy consumption, as a function of the tractor speed, was developed. Considering the dependency of the mechanical required power on the speed, the following relation can be written:

$$P \propto P_1 \cdot V^3 + P_2 \cdot V \tag{2.9}$$

where P_1 and P_2 are defined in Eq. 2.10 and Eq. 2.11 respectively, for the case in which the tractor is not towing any aircraft. Analogous equations can be derived for the towing case.

$$P_1 = \frac{1}{2} \cdot \rho \cdot A \cdot C_X \tag{2.10}$$

$$P_2 = m_t \cdot g \cdot f_r \tag{2.11}$$

Considering a constant speed along the whole segment, it is possible to find the total traveling time as $\Delta t_{tot} \simeq d/V$, where d is the total distance discounted by the acceleration and deceleration segments. Thus, from Eq. 2.7, where ΔE_{acc} and ΔE_{dec} represent the energy consumed to accelerate and decelerate obtained from the aforementioned database, it is possible to compute the total consumed energy as:

$$\begin{aligned} \Delta E_{tot} &= \frac{1}{\eta_{out} \cdot \eta_d} \cdot \left(\frac{P_1 \cdot V^3 + P_2 \cdot V}{\eta_{EM}} + P_{aux} \right) \cdot \frac{d}{V \cdot 3600} + \Delta E_{acc} + \Delta E_{dec} = \\ &= \frac{d}{\eta_{out} \cdot \eta_d \cdot 3600} \cdot \left(\frac{P_1 \cdot V^2 + P_2}{\eta_{EM}} + P_{aux} \cdot V^{-1} \right) + \Delta E_{acc} + \Delta E_{dec} \end{aligned} \tag{2.12}$$

This consumption model grants a fast computation of the energy consumed by the tractor to travel along a taxiway segment and good accuracy, as shown by the black line in Fig. 2.4; thus, it can be effectively implemented within an optimization scheme, preserving its computational time performance.

It must be noticed that, in the proposed model, the static friction (from standstill to moving) is neglected. In fact, even though the amount of torque required to overcome the static friction is considerably higher than the one required to win the rolling friction and must be taken into consideration when designing the tractor engine, the static friction can be considered as a spike [94], thereby leading to $\Delta t \to 0$ and $\Delta E \to 0$.

2.4 Trajectory assignment problem formulation

The trajectory assignment problem (TAP) is defined as a combinatorial problem and consists of finding the set of non conflicting taxiing trajectories for each tractor mission characterized by the minimum cost. The taxi speed is modeled as a discrete variable: $\mathbf{V} = \{V_{min}, V_{min} + \Delta V, \ldots, V_{max}\}^T$, with minimum and maximum speed $V_{min} = 4\ m/s$ and $V_{max} = 16\ m/s$ and speed step $\Delta V = 0.5$. This speed range has been chosen to meet the maximum taxi speed achievable using state of the art

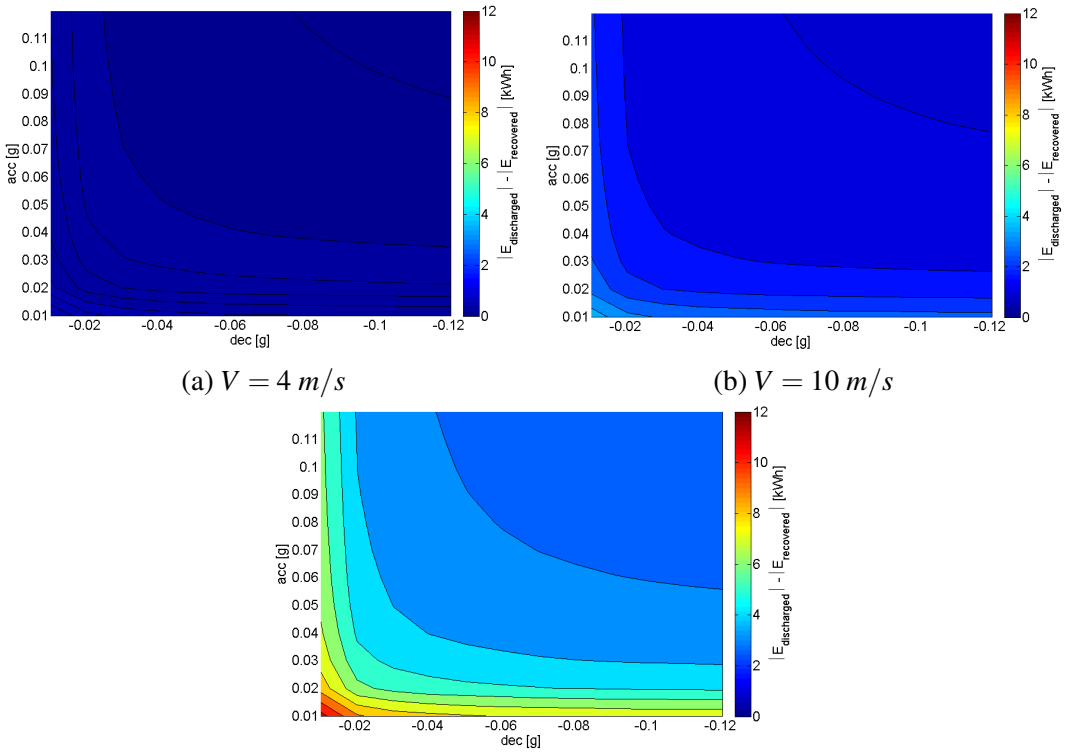
pushback tractors. The set of trajectories $\chi \in \mathbf{X}$, corresponding to a solution of the problem, is defined as follows:

$$\begin{aligned}\chi = &\{\left(p_j, V_j, PB_{buff}\right)_k \mid p_j \in \mathbf{P}_k, V_j \in \mathbf{V}, \\ &PB_{buff} \in \mathbf{buff}, j = \{1,2,3\}, k = 1,\ldots,NF\}\end{aligned} \tag{2.13}$$

where $\left(p_j, V_j, PB_{buff}\right)_k$ represents the set of path, speed and pushback buffer time assigned to the j-th phase of the k-th mission, $p_j \in \mathbf{P}_k$ is the index of the path, among the k-shortest computed paths, selected for the phase j of the considered mission k (see Section 2.2) and NF is the total number of departures/arrivals. The set of possible paths for all the phases of the missions is $\mathbf{P} = \bigcup \mathbf{P}_k$. The buffer time is modeled as a discrete variable as well, with $\mathbf{buff} = \{0, \Delta buff, \ldots, MaxBuffer\}^T$ defined as the set of possible buffer time, with a time step of $\Delta buff = 1s$. For landing flights the variable PB_{buff} is constrained to 0.

In order to detect possible conflicts between the tractors moving on the airport surface, the algorithm must keep track of the movements of the tractors. Two models, one considering a continuous time evolution and one based on discrete time formulation, were studied. After an analysis of the consumption model proposed in Section 2.3, some considerations can be made on the path deconfliction strategy. The energy spent in accelerating to a traveling speed V can not be fully recovered while braking, due to the coefficient e_{rec} and to the lower recharge efficiency of the electric system, with respect to the discharge case. This feature is depicted in Fig. 2.6 for three speed levels: $V = V_{min}$, $V = 10\ m/s$ and $V = V_{max}$. The figures show the net energy consumption in the acceleration and deceleration segments, for different combinations of the acceleration and deceleration values, which were varied from 0.01 g to 0.12 g.

Hence, in the proposed approach, operations are conceived as continuous, which means that the tractors never stop during the motion between two points within the airport. However, this strategy represents a highly restrictive constraint for the trajectory assignment problem. Therefore, the tractor is allowed to spend a buffer time at the end of the first phase (buff 1) and at the beginning of the third one (buff 2), as sketched in Fig. 2.7. The buffer upper limit is set to $MaxBuffer = 10\ min$. During the buffer time, the tractor is assumed to be in idle, thereby reducing the electrical



(a) $V = 4\ m/s$ (b) $V = 10\ m/s$

(c) $V = 16\ m/s$

Fig. 2.6 Net energy consumption between acceleration and deceleration phases $E_{discharge} - E_{recovered}$.

power consumption to account only to the main onboard computer ($P_{aux_{idle}} = 0.2\ kW$). In this case, the maximum energy consumption is quantified in 0.072 kWh, which is one order of magnitude lower than the smallest amount of energy lost with an acceleration/deceleration cycle.

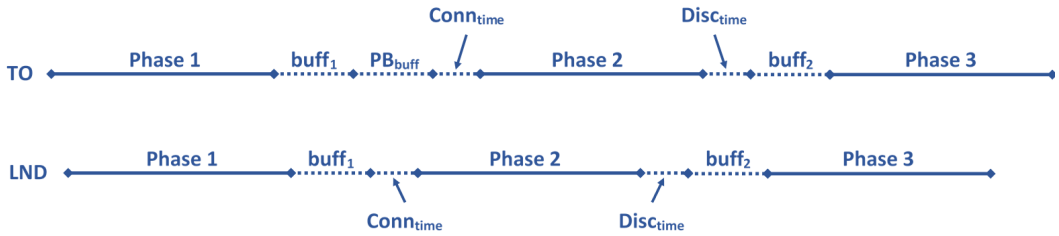


Fig. 2.7 Mission representation including buffer times.

Given the introduction of the buffer time variables $buff_1$ and $buff_2$, the generic solution $\chi \in \mathbf{X}$ of the trajectory assignment problem assumes the form reported in Eq. 2.14.

$$\chi = \{ \left(p_j, V_j, PB_{buff}, buff_1, buff_2\right)_k \mid p_j \in \mathbf{P}_k, V_j \in \mathbf{V}, \\ PB_{buff} \in \mathbf{buff}, buff_1 \in \mathbf{buff}, buff_2 \in \mathbf{buff}, j = \{1,2,3\}, k = 1,\ldots,NF\} \tag{2.14}$$

2.4.1 Continuous time based model

As far as the continuous time model is considered, for each taxiway section, the algorithm computes the time at which the tractor enters the taxiway segment and the time at which it leaves it, including acceleration/deceleration if part of the considered segment; this data are collected in matrix form, as showed in Fig 2.8. Four types of conflict can occur (Fig. 2.9): tail-head on the same segment, tail-head on consecutive segments, head-head on the same segment and head-head on contiguous segments.

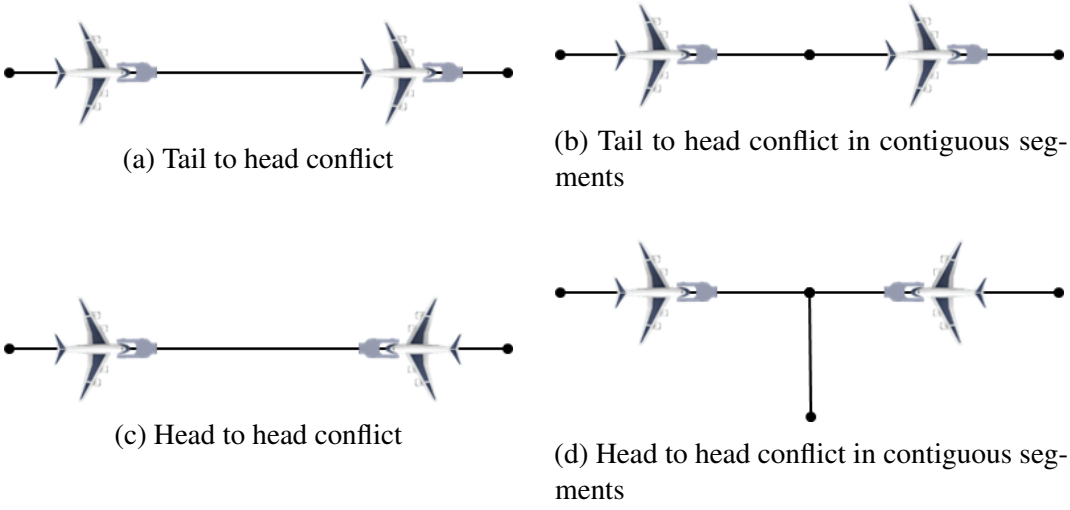
Segment	Starting node	Ending node	Starting time	Ending time
1	102	101	64371.63	64377.79
2	101	100	64377.79	64391.91
3	100	30	64391.91	64415.04
4	30	32	64415.04	64427.30
5	32	80	64427.30	64439.22
6	80	33	64439.22	64490.09
7	33	20	64490.09	64551.50
8	20	19	64551.50	64673.30
9	19	78	64673.30	64717.70

Fig. 2.8 Continuous time model scheduling matrix form.

A tail-head conflict (Fig. 2.9a - 2.9b) is detected if two tractors enter or leave a segment with time separation lower than the *safety_gap*, which take on one of two forms depending on the taxiing speed: the first term considers the dimensions of tractor and towed airplane, the second one is the time required to stop (Eq. 2.15).

$$safety\_gap = \begin{cases} \left(\frac{l_{airplane}}{V_f} + \frac{V_f}{a}\right) \cdot sf & if\ towing \\ \left(\frac{l_{tractor}}{V_f} + \frac{V_f}{a}\right) \cdot sf & otherwise \end{cases} \tag{2.15}$$

where V_f is the taxi speed of the follower, a is the deceleration, $l_{airplane}$ and $l_{tractor}$ are the trailing airplane and tractor lengths respectively and $sf = 1.1$ is a safety factor.



(a) Tail to head conflict

(b) Tail to head conflict in contiguous segments

(c) Head to head conflict

(d) Head to head conflict in contiguous segments

Fig. 2.9 Conflict types for the continuous time model.

This formulation takes into account also the dimensions of the moving object and avoids any part of the vehicles getting in contact.

In order to evaluate if a head-head conflict on two contiguous segments (Fig. 2.9d) can occur, the separation at the end node of the respective segments is compared to the threshold described in Eq. 2.16, with V_1 and V_2 representing the taxi speed of the two tractors.

$$safety\_gap = \begin{cases} \max\left(\left(\frac{l_{airplane}}{V_1} + \frac{V_1}{a}\right), \left(\frac{l_{airplane}}{V_2} + \frac{V_2}{a}\right)\right) \cdot sf & if\ towing \\ \max\left(\left(\frac{l_{tractor}}{V_1} + \frac{V_1}{a}\right), \left(\frac{l_{tractor}}{V_2} + \frac{V_2}{a}\right)\right) \cdot sf & otherwise \end{cases} \tag{2.16}$$

Finally, if at any time two tractors are traveling at the same time on the same taxiway segment in opposite direction, a head-head conflict (Fig. 2.9c) is raised.

2.4.2 Discrete time model

The discrete time model uses a built-in simulator to evaluate the position of each tractor at a generic time t. Following the approach proposed in [95], the planning horizon has been discretized with a time step of $\Delta t = 5s$. The size of the time step

was chosen based on the trade-off between the computational resources required to simulate the entire horizon and the effectiveness in finding possible conflicts. The time spent by a tractor on a specific taxiway segment is expressed in periods, as reported in Eq. 2.17.

$$periods = \frac{d}{V \cdot \Delta t} \tag{2.17}$$

The resulting value is rounded to the upper integer, thereby adding a buffer on the time spent on a segment; however, in certain cases, the traveling time on a segment can be excessively overestimated, resulting in a potentially erroneous conflict detection.

In order to detect possible conflicts between the tractors moving on the airport surface, a rectangular Threat Detection Area (TDA) is defined around each tractor, with the long edge aligned to the taxi speed (Fig. 2.10). If round TDAs were to be considered around each tractor, the case of two tractors traveling on parallel taxiways might be erroneously identified as a conflict. The choice of a rectangular shape avoids this behavior.

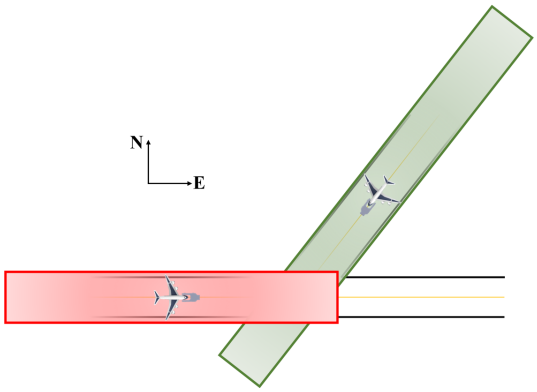


Fig. 2.10 Threat detection area scheme.

The TDA equation must take into consideration the actual tractor position in longitude and latitude (λ_o, φ_o) and the heading angle ψ assumed by the tractor, while traveling along the different taxiways; the mathematical definition of the TDA is reported in Eq. 2.18, where $\theta = \pi/2 - \psi$.

$$\begin{aligned} TDA = &\{\{\lambda', \varphi'\} \mid \lambda' = \lambda_o + (\lambda - \lambda_o) \cdot \cos\theta - (\varphi - \varphi_o) \cdot \sin\theta \\ &\varphi' = \varphi_o + (\varphi - \varphi_o) \cdot \cos\theta + (\lambda - \lambda_o) \cdot \sin\theta, -\frac{c}{2} \leq \lambda \leq \frac{c}{2}, -\frac{b}{2} \leq \varphi \leq \frac{b}{2}\} \end{aligned} \tag{2.18}$$

The short edge b of the TDA, perpendicular to the taxiway centerline, has constant length defined in Eq. 2.19.

$$b = \begin{cases} \left(w_{airplane} + \varepsilon\right) \cdot \dfrac{sf}{R_E} & \textit{if towing} \\ \\ w_{tractor} \cdot \dfrac{sf}{R_E} & \textit{otherwise} \end{cases} \tag{2.19}$$

The parameters $w_{tractor}$ and $w_{airplane}$ are respectively the tractor width and the airplane wingspan; $\varepsilon = 7.5m$ was taken as the maximum clearance between a parked aircraft and any other object in the airport, as defined by the ICAO [93]. $R_E = 6371km$ is the Earth radius, used to convert the TDA size in latitude and longitude variations. A safety factor $sf = 1.1$ was also introduced.

The long edge c was conceived to allow two tractors, with potential conflicting paths, to stop without getting in contact. The expression for c has two components: a constant term depending on the tractor/airplane length, and a variable term depending on the tractor speed (Eq. 2.20).

$$c = \begin{cases} \left(l_{airplane} + \dfrac{V^2}{2 \cdot a}\right) \cdot \dfrac{sf}{R_E} & \textit{if towing} \\ \\ \left(l_{tractor} + \dfrac{V^2}{2 \cdot a}\right) \cdot \dfrac{sf}{R_E} & \textit{otherwise} \end{cases} \tag{2.20}$$

To detect whether, at a certain time $t \in \mathbf{T}$, the TDAs of two tractors overlap, a test based on the separating axis theorem was implemented [96–98]. Given the axes perpendicular to the edges of one of the TDAs, consider the projection of the vertices of the polygons on the axes (Fig. 2.11); if there is no overlap between the projections on at least one of the axes, the two TDAs are separated.

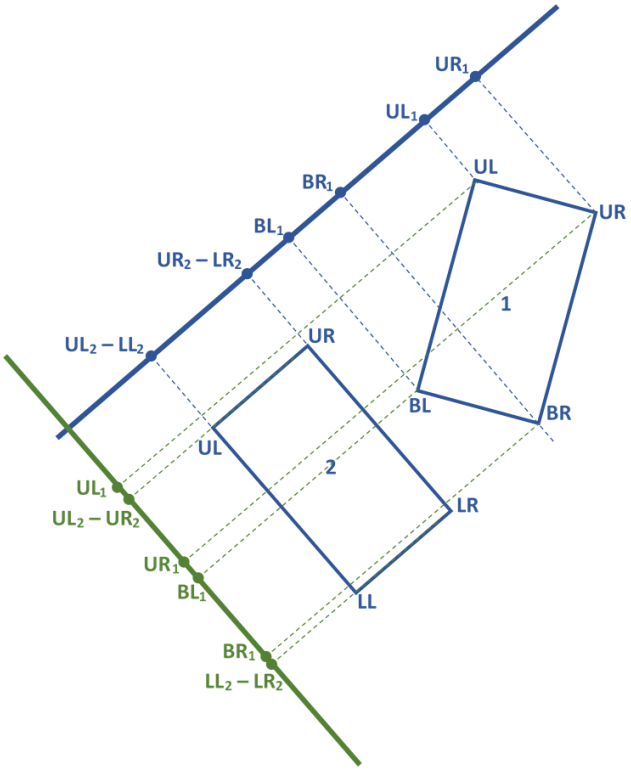


Fig. 2.11 Separating axis theorem scheme.

2.4.3 Mathematical problem formulation

The objective of the trajectory assignment algorithm is to minimize the total cost of taxi operations (Eq. 2.21). The expression for the cost has two terms: the total energy consumption, defined by the variable C_E $[kWh]$ which ignores the own powered segment consumption, and the total buffer time C_T $[s]$. The weighting coefficient $k_t = 0.01$ is used to balance the order of magnitude of the two terms. The mathematical formulation of the trajectory assignment problem is reported hereinafter:

$$\text{Minimize} \quad \sum_{i=1}^{NF} (C_{E,i} + k_t \cdot C_{T,i}) \tag{2.21}$$

Subject to the static constraints

$$\begin{cases} separation > safety\_gap & continuous\ time \\ TDA_{t,i} \cap TDA_{t,j} = \emptyset \qquad \forall t \in \mathbf{T}, \quad i,j = 1,\ldots,NF, \quad i \neq j & discrete\ time \end{cases} \tag{2.22}$$

$$t_{PB,i} - t_{A,i} \geq TAT_i \qquad i = 1,\ldots,NF \tag{2.23}$$

$$t^a_{A,k} - t^p_{PB,k} \geq MSG \qquad \forall k \in \mathbf{G} \tag{2.24}$$

$$t^a_{PB,k} - t^a_{A,k} \leq MGO \qquad \forall k \in \mathbf{G} \tag{2.25}$$

$$t^a_{RWY,i} - t^p_{RWY,j} \geq sep_{min} \qquad i,j = 1,\ldots,NF \tag{2.26}$$

$$V_{i,j} \in \mathbf{V} \qquad i = 1,\ldots,NF, \quad j = \{1,2,3\} \tag{2.27}$$

$$p_{i,j} \in \mathbf{P} \qquad i = 1,\ldots,NF, \quad j = \{1,2,3\} \tag{2.28}$$

$$0 \leq PB_{Buff,i} \leq MaxBuffer \qquad i \in \mathbf{TO} \tag{2.29}$$

$$0 \leq buff_{i,j} \leq MaxBuffer \qquad i = \{1,2\}, \quad j = 1,\ldots,NF \tag{2.30}$$

The path deconfliction is ensured by Eq. 2.22 for both continuous and discrete time models. Equations 2.23-2.25 represent the stand occupation constraints, with **G** being the set of the aircraft stands. Before an aircraft can leave its parking spot, several operations must be performed; the minimum time required to complete these operations is called minimum turnaround time (TAT) and it is specific of each aircraft type and depends also on the flight length (i.e. domestic, international). Moreover, airports define a minimum separation between two aircraft at the stand

Table 2.2 Aircraft wake turbulence classification.

Class	MTOW range [t]
Small	≤ 18
Large	$18 < MTOW < 136$
Heavy	≥ 136

Table 2.3 Wake turbulence separation minima.

Following aircraft	Leading aircraft	separation [min]
Heavy	Heavy	2
Large	Heavy	2
Small	Heavy	3
Small	Large	3

$MSG = 10\ min$ (Eq. 2.24) to allow the operators get ready for the following flight and a maximum presence for each airplane $MGO = 2\ h$ (Eq. 2.25). The variables t_{PB} and t_A are respectively the pushback time and the arrival time at the parking lot. The superscript p represents the previous airplane that was parked at the lot, whereas a indicates the actual airplane. The aircraft sequencing at the runway is constrained by the minimum separation requirements set by the ICAO and FAA [99]. The main factor affecting the runway separation is the wake turbulence generated by an aircraft during landing or takeoff: the wake generated at the wing tip can interfere with the aerodynamics of the following aircraft, thereby causing potential hazards. This phenomenon is emphasized when the leading aircraft is bigger than the follower. Three main categories of airplanes are defined, depending on the maximum takeoff weight: small, large and heavy (Table 2.2) [99]. In the proposed mathematical model, a correct runway separation is enforced by Eq. 2.26, where t^a_{RWY} is the time at which the actual considered aircraft starts the takeoff roll or lands and t^p_{RWY} is the analogous time referred to the preceding aircraft. The values of the separation minima sep_{min} are summarized in Table 2.3, in the case of same runway operations [99].

The variable bounds are defined by Eq. 2.27-2.30 for V, p, PB_{Buff} and *buff* respectively. The set **TO** is the subset of the flight schedule that contains the sole departures, where NTO is number of departures.

2.4.4 Computational complexity

The computational complexity analysis of the trajectory assignment problem is based on two main concepts: *NP*-hard problem and self-reducibility. The informal definitions of these two concepts can be stated as follows [100]:

Definition 2.1 (Deterministic Turing machine)**.** *A deterministic Turing machine is a mathematical model that defines an abstract machine, designed to manipulate symbols on a strip of tape, according to a table of rules.*

Definition 2.2 (Non-deterministic Turing machine)**.** *A non-deterministic Turing machine is a Turing machine, which set of rules prescribes more than one action for a given situation.*

Definition 2.3 (*P* complexity class)**.** *A decision problem is part of the P class if it can be solved by a deterministic Turin machine in a polynomial time*

Definition 2.4 (*NP* complexity class)**.** *A decision problem is said to be NP if it can be solved by a non-deterministic Turin machine in a polynomial time*

Definition 2.5 (*NP*-hard problem)**.** *A search problem* Π *is said to be NP-hard if it some NP-complete decision problem* Π' *exists, such that it Turing-reduces to* Π*. It is indicated with* $\Pi' \propto_T \Pi$*.*

Definition 2.6 (Self-reducibility)**.** *A search problem* Π *is self-reducible if the corresponding decision problem* $\Pi' \in NP$ *Turing-reduces to* Π*. That is,* Π *is self-reducible if there is a polynomial-time Turing machine to solve* Π *that calls an oracle subroutine S that solves* Π'*.*

The trajectory assignment problem can be proved to be *NP*-hard, thereby intractable, which mathematically means that it cannot be solved in polynomial time unless $P = NP$. The self-reducibility technique is used to prove the *NP* hardness of the optimization problem (search problem) [100, 101]. The corresponding decision problem will be proved to be *NP*-complete; subsequently, a polynomial-time Turing

reduction (Cook reduction) will be outlined to prove that TAP is self-reducible. The notation used in [100] will be adopted in the following discussion. The trajectory assignment decision problem can be stated as follows:

Instance: The inputs of the trajectory assignment decision problem are:

- a flight schedule and a set of taxiway segments E;
- for each arrival and departure $1 \leq j \leq NF$:
  - a set of three paths $Ps = \{p_1(j), p_2(j), p_3(j)\} \subset E$;
  - the corresponding set of taxiing speeds $Vs = \{V_1(j), V_2(j), V_3(j)\} \in \mathbf{V}$;
  - a set of three buffer times $Buffers = \{buff_1(j), buff_2(j), PB_{buff}(j)\} \in \mathbf{buff}$;
- for each path k:
  - a path length $d_k(j) \in \mathbb{R}_0^+$;
  - a corresponding time period $l_k(j, V_k(j)) \in \mathbb{R}^+$;
  - an earliest starting time $et_k(j) \in \mathbb{R}^+$;
  - a deadline time $dt_k(j) \in \mathbb{R}^+$; a cost $c_k(j, V_k(i)) \in \mathbb{R}$;
- a constant $K \in \mathbb{R}^+$.

Question: Is there a set of paths, speeds and buffer time that generates schedules for the taxi operations without conflicts between the trajectories, respecting the runway separation minima, and with total cost $\sum_{j=1}^{NF} \left(\sum_{k=1}^{3} c_k(j) + k_t \cdot (buff_1(j) + buff_2(j) + PB_{buff}(j)) \right) \leq K$? ■

Before showing the proof of the *NP*-completeness of the trajectory assignment decision problem, we need to recall a problem that is known to be *NP*-complete: the *no-wait job-shop* scheduling, to which the trajectory assignment decision problem will be restricted.

Instance: The inputs of the trajectory assignment decision problem are:

- the number $m \in \mathbb{Z}^+$ of processors;
- a set of jobs J, each $j \in J$ consisting of an ordered collection of tasks $t_k \in j$ $1 \leq k \leq n_j$;
- for each task $t \in j$:
  - a length $l_t \in \mathbb{Z}_0^+$;

- a processor $p_t \in \{1,2,\ldots,m\}$, where $p_{t_k} \neq p_{t_{k+1}}$, with t_k and $t_{k+1} \in j$, for all $j \in J$ and $1 \leq k \leq n_j$
- a deadline $D \in \mathbb{Z}^+$.

Question: Is there a *job-shop* schedule for J that meets the overall deadline, namely a collection of one-processor schedules σ_i with $1 \leq i \leq m$, such that:

- for two tasks of different jobs t and t', $\sigma_{i,t} > \sigma_{i,t'}$ implies $\sigma_{i,t} \geq \sigma_{i,t'} + l_{t'}$;
- for two tasks of the same job t_{k+1} and t_k, $\sigma_{t_{k+1}} = \sigma_{t_k} + l_{t_k}$, with t_k and $t_{k+1} \in j$;
- $\sigma_{t_{n_j}} + l_{t_{n_j}} \leq D$;

for all $j \in J$ and $1 \leq k \leq n_j$? ■

The *no-wait job-shop* scheduling can be easily proved to be *NP*-complete, showing that for $m = 1$ it restricts to the *traveling salesman problem*, also known to be *NP*-complete [100]. It is possible now to state Theorem 2.1.

Theorem 2.1 (TAP decision *NP*-completeness)**.** *The trajectory assignment decision problem is NP-complete.*

Proof. The decision problem can be restricted to *no-wait job-shop* scheduling, with the paths representing the jobs $j \in J$, the path segments as the tasks $t_k \in j$ and the taxiways E being the processors. It is done by allowing only instances with Ps containing one path for each phase of each mission, buffer time equal to zero, constant taxiing speeds V', thereby implying a fixed task length $l'_k \in \mathbb{R}^+$, earliest starting times $et_k = 0$, deadlines $dt_k = D$, cost $c_k = l_k$ and having $K = 3 \cdot NF \cdot D$. ■

The second step of the proposed proof of intractability for the trajectory assignment problem consists in proving the self-reducibility of the problem, to demonstrate Theorem 2.2.

Theorem 2.2 (TAP *NP*-hardness)**.** *The trajectory assignment problem is NP-hard.*

Proof. Suppose that $S[x,K]$ is a subroutine to solve the TAP decision problem for a generic instance x, and that the optimal cost is denoted by K^*. Each instance x has $n = 9 \cdot NF$ variables that can be ordered in a solution vector $\phi = (\phi_1, \ldots \phi_n)$ that contains, for all $1 \leq j \leq NF$, respectively the paths p_k, the speeds V_k, with $1 \leq k \leq 3$, and the buffer times $buff_1$, $buff_2$ and PB_{buff}.

We know that the optimal cost lies between the values $K_{min} = K(Ps_{min}, Vs_{min}, Buffers_{min})$ and $K_{max} = K(Ps_{max}, Vs_{max}, Buffers_{max})$, with the parameters standing for $Ps_{min} = \{p_{1,shortest}, p_{2,shortest}, p_{3,shortest}\}_j$, $Vs_{min} = \{V_{min}\}^{NF}$, $Buffers_{min} = \{0\}^{NF}$, $Ps_{max} = \{p_{1,longest}, p_{2,longest}, p_{3,longest}\}_j$, $Vs_{max} = \{V_{max}\}^{NF}$, $Buffers_{max} = \{MaxBuffer\}^{NF}$, for $1 \leq j \leq NF$. Therefore, it is possible to determine the value K^* using a binary search procedure that calls the subroutine $S[x,K]$ with different values of K. It can be considered, without loss of generality, that the number of paths available for each phase of each mission is equal and defined as Np. Therefore, the binary search runs at most in $\lceil \log_2 \left(NF^9 \cdot Np^3 \cdot NV^3 \cdot Nb^3\right) \rceil$ iterations.

Once the optimal cost K^* has been computed, the following procedure can be applied to find the solution vector ϕ^* that gives a conflict-free schedule with total cost K^*. It must be noticed that, if more than one set of variable values solve the optimization problem, they are considered equivalent; therefore, it is sufficient that the algorithm outputs any of these sets.

- For $j = 1$ to n do:
  - Substitute ϕ_j with one of its possible values ϕ'_j and run the subroutine $S[x', K^*]$.
  - If the modified ϕ is not satisfiable, substitute ϕ_j with its subsequent possible value.
  - Repeat the two steps above until a satisfiable ϕ is found, thereby fixing $\phi_j = \phi^*_j$.
- Output $\phi^*_1, \ldots, \phi^*_n$.

The above algorithm, when used to solve the search problem, given a subroutine S for the correspondent decision problem, runs in maximum $3 \cdot NF \cdot (NP + NV + Nb) = n/3 \cdot (NP + NV + Nb)$ iterations, beyond the iterations required to find K^*. Therefore, the algorithm runs in polynomial time, thereby constituting a polynomial-time Turing reduction and proving that TAP is self-reducible. As the TAP decision problem is NP-complete, it descends that the corresponding optimization problem is NP-hard. ■

2.5 Tractor dispatch problem formulation

The tractor dispatcher aims to find the optimal assignment of tasks (departures/arrivals) to resources (tractors), with the final objective of maximizing the resources utilization. In this application, the number of tractors NT can be lower than the tasks to be accomplished; moreover, the tractor battery has a limited capacity. Therefore, not all the tractors might always be available.

The state of charge of a tractor at the time t is computed using Eq. 2.8. The maximum value of the depth of discharge (DoD), with $DoD = 1 - SoC$, is set to $maxDoD = 0.7$, as a trade-off between the tractor availability and the battery cycle life. When the battery level goes below the *maxDoD*, the tractor recharges it in the charging stations placed at the depot. Actual trends in the performance of the charging stations set the charging time of a 90 kWh battery to 40 *min* to reach 80% of the charge and to 75 *min* for the full charge starting from complete discharge condition [102]. Hence, in the present application, the time to charge (TtC) for tractors with characteristics presented in Table 2.1 was set to $TtC = 30$ *min*.

2.5.1 Mathematical problem formulation

The tractor utilization U is defined as the ratio between the tractor operating time t_{oper}, namely the sum of traveling, buffer, and charging times, and the computational horizon time (Eq. 2.31).

$$U = \frac{t_{oper}}{t_h} \tag{2.31}$$

In order to maximize the utilization of each tractor, the standard deviation σ_U among the whole fleet is minimized, as described in Eq. 2.32, with $\bar{U}$ being the average utilization. This criterion was selected as optimization driver in order to minimize vehicle downtime, which represents a cost for the ground handling company.

$$\text{Minimize} \quad \sigma_U = \sum_{i=1}^{NT} \frac{1}{NT} \cdot (U_i - \bar{U})^2 \tag{2.32}$$

Subject to

$$\sum_{m=1}^{NF} job_{t,i,m} \leq 1 \qquad \forall t \in \mathbf{T}, \quad i = 1,\ldots,NT \tag{2.33}$$

$$SoC_{t,i} > 0 \qquad \forall t \in \mathbf{T}, \quad i = 1,\ldots,NT \tag{2.34}$$

Each tractor can be assigned to only one flight at a time; therefore, the binary variable $job \in \{0,1\}$ has been introduced in Eq. 2.33. The tractor schedule is ensured to be congruent with the battery capacity by Eq. 2.34, which bounds the depth of discharge of the battery.

2.5.2 Computational complexity

The tug dispatch problem can be thought as an asymmetric *multi-trip multi-vehicle routing problem* with time windows (AMVRPTW). In the proposed application, vehicles do not have a load constraint, but an energy consumption constraint is instead defined. The concepts of trip and journey and the formulation of the *multi-trip multi-vehicle routing problem* with time windows are defined hereinafter [103].

Definition 2.7 (Trip)**.** *A trip is a sequence of served customers that starts and ends at the depot.*

Definition 2.8 (Journey)**.** *A journey is a set of one or more trips.*

Instance: A directed graph $G = (V,A)$, where $V = \{V_0, V_1, \ldots, V_{NF}\}$ is the set of vertices and $A = \{(V_i, V_j) \mid V_i, V_j \in V,\ V_i \neq V_j\}$ the set of arcs. Arcs (V_i, V_j) are characterized by their travel time T_{V_i,V_j}. The first node V_0 represents the depot where the fleet of identical vehicles v with maximum energy capacity Q is available at time 0 and has to return at time T_H. Each vertex corresponds to a customer to be visited by a vehicle, requiring a certain quantity of product $Q_i \in \mathbb{R}^+$. A time window $[E_{V_i}, L_{V_i}]$ is defined for each customer; the vehicle is allowed to arrive at the customers node before the time window starts, thereby defining a waiting time. However, if the tractor reaches the customer after the conclusion of the corresponding time window, the trip is considered infeasible.

Question: Is there a journey assigned to each vehicle that respects the following characteristics:

- each trip start and ends at V_0;
- $\sum_i Q_i \leq Q$ for each customer $1 \leq i \leq n_c$ in a single trip;
- customer time windows are respected;
- each customer is visited exactly once;
- trips assigned to the same vehicle do not overlap;
- the duration of the journey of each vehicle is less than T_H;

and that minimizes the traveled time? ■

The formulation of the tug dispatch problem as an AMVRPTW is reported hereinafter.

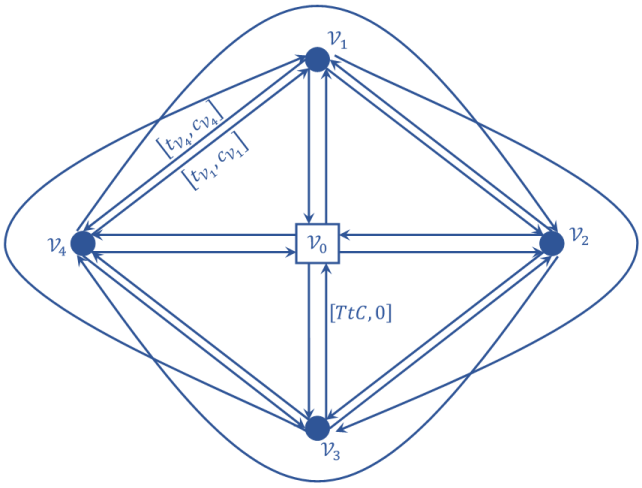


Fig. 2.12 Scheme of the vehicle routing problem model modeling.

Let $G = (V,A)$ be a directed digraph with different weights for the two arcs connecting two nodes (Fig. 2.12), where the vertices V represent the tractor missions. Each arc (V_i, V_j) is characterized by time travel and a cost $T_{V_i,V_j} = [t_{V_j}, c_{V_j}]$, which are defined respectively as the time required to perform the mission j and the associated cost. The first node V_0 represents the charging station for the fleet of identical tugs $v = \{v_1, \ldots, v_{NT}\}$ with maximum energy capacity $Q = 1$, where the tractors are available at time 0 and have to return when no more missions have to be performed. When a vehicle completes a mission and its total cost is above *maxDoD*, it has to return to the charging station, where the cost level will be set back to 0. The time travel and costs associated to the arcs arriving at the charging station are $T_{V_i,V_0} = [TtC, 0]$. Each vehicle is allowed to perform more than one trip. For the tug dispatch problem, the bounds of the time windows coincide and are equal to the end

of the mission $E_{V_i} = L_{V_i} = end_{V_i}$ for $1 \leq i \leq NF$. A vehicle is allowed to arrive at the nodes before the assigned time windows; however, a waiting time is added until the time window opens. This waiting time is considered as the time that the tugs spend at the depot between missions. Given the previous definition of time window, the completion time T_H loses its meaning, as it would be constrained by the end of the last mission to be accomplished. The objective is to maximize the operative time for each tractor, that means to minimize the standard deviation of the utilization among the fleet, while ensuring that the total cost of a trip is always lower than the capacity Q.

Theorem 2.3 (Tug dispatch *NP*-hardness)**.** *The tug dispatch problem is NP-hard.*

Proof. The tug dispatch problem formulation reported above is analogous to the *multi-trip multi-vehicle routing problem* with time windows, which has been proved to be *NP*-hard [103]. In fact, the dispatch problem can be restricted to the vehicle routing problem, which is *NP*-hard. The definition of time windows complicates the research of a solution, as a modification of one trip affects the vehicle journey [103]. This feature is stressed when the constraint imposed by the time windows is reinforced by equating the two extremes of the time window. ■

Chapter 3

Conflict-free ground routing and tug dispatch algorithms

In the last decade, the airside ground routing problem has gained more attention among researchers, due to increasing airside traffic of airports, i.e. the parts of the airport with aircraft access to the runways. Moreover, the objective of reducing the environmental impact and the costs of airport operations has led many in the aeronautical community to study and develop new solutions to established airport procedures. The enhanced capabilities of self-driving vehicles have supported the development of solutions that use autonomous or semi-autonomous tugs to perform the pushback or to tow the aircraft between stands and runways; the work presented in [31, 33, 32, 35] are examples of these solutions. Although several studies about the feasibility and the advantages brought by these solutions have been proposed, the literature lacks studies about the operative management of the fleet of autonomous tugs.

The derivation and implementation of three optimization algorithms is presented in this chapter for the conflict-free ground routing problem (Section 3.1): two hybridization of the particle swarm optimization (PSO) and a tree-search heuristic are proposed to solve the trajectory assignment and departure sequencing problems studied in Section 2.4. The implementation of the hybrid particle swarm optimization (HPSO) algorithms for the tug dispatch problem is introduced in Section 3.2.

3.1 Trajectory assignment and departure sequencing algorithm

The mathematical formulation of the trajectory assignment problem presented in Section 2.4 uses discrete variables to represent the speed and possible paths for each mission and the time buffers. Thus, the optimal set of non-conflicting trajectories can be found by solving a combinatorial optimization problem. The number of possible combinations of paths, speeds and buffer time for all the tractor missions is computed by means of Eq. 3.1, where $Np_{i,j}$ is the number of possible paths for phase j of mission i, NV represents the number of possible speed values and Nb is the size of the set of buffer times.

$$combinations = \prod_{i=1}^{NF} \left(\left(\prod_{j=1}^{phases} \left(Np_{i,j} \cdot NV \right) \right) \cdot \begin{cases} \left(Nb^3 \right) & if\ i \in \textbf{TOs} \\ \\ \left(Nb^2 \right) & otherwise \end{cases} \right) \tag{3.1}$$

After the analysis of several test cases, the search domain of the present problem has shown to be most likely non-convex, as illustrated in Fig. 3.1, where the feasible domain is depicted for two test cases composed of three flights each and with the only towing phase considered. Therefore, linear programming techniques (e.g. Simplex) cannot be used. Moreover, the high complexity of the problem highlighted in the previous chapter led to the choice of using approximate methods, to find an optimal or near-optimal solution in a limited computational time.

3.1.1 Hybrid particle swarm optimization algorithm

Two hybrid versions of the PSO were proposed, based on a hill-climbing algorithm and on an algorithm derived from variable neighborhood search techniques. At the end of each iteration of the PSO, the local search algorithm runs to improve the position of each particle (Algorithm 3 line 9). An algorithm to predict the best value for each element of the particle was also developed. It runs after the hybridization algorithm to determine which value of each particle element has the highest probability of being part of the optimal solution (Algorithm 3 line 10).

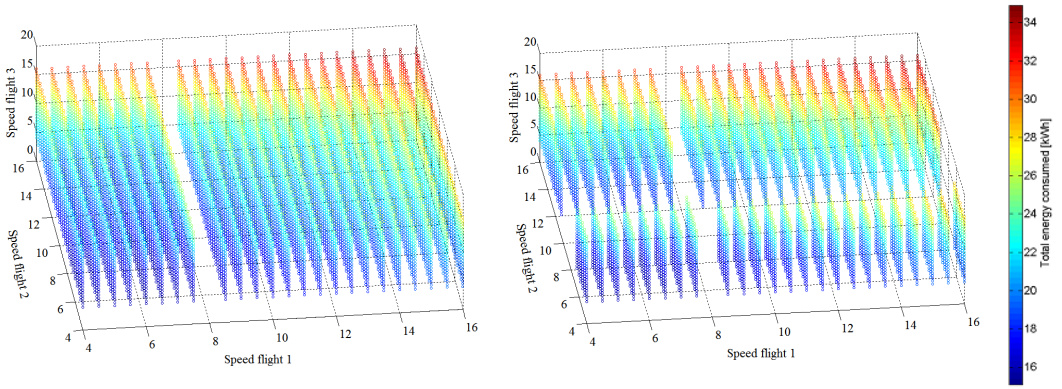


Fig. 3.1 Feasible search domain for two test cases with three flights each and only the towing phase considered.

Algorithm 3 Hybrid particle swarm optimization with variable fixing pseudocode.

```
Generate K random particles
Initialize velocities v_{k,j} between 0 and 1
while (Stopping condition) = FALSE do
    Evaluate the fitness for each particle
    Determine the best solution pbest visited so far
    Determine the global best solution gbest visited so far
    Compute the particle velocities
    Update the particle position
    Improve particle positions with hybridization
    Run variable fixing algorithm
```

3.1.1.1 HC-HPSO algorithm

Hill-climbing is a local search meta-heuristic that is used to exploit the particle position and, thus, to improve the PSO solution at each iteration. For every particle, m integer numbers are randomly extracted from the $\{1,\ldots,n\}$ set; these integers represent the indexes of the elements of the particles to be analyzed by the hill-climbing. The value of each examined element is made vary within the defined range for that element, while the remaining $n-1$ elements are fixed. The fitness value is evaluated for each modified particle and the one characterized by the maximum fitness value replaces the original particle (Algorithm 4). Therefore, with respect to the classical hill-climbing heuristic, two elements of stochasticity were introduced: the particle element selection process and the order in which they are analyzed; this grants a broader search of the space around the actual particle position, without a

predefined search direction. The number of elements to be modified by the hill-climbing heuristic was set after a parameter tuning, which will be presented in Section 3.1.1.7.

Algorithm 4 Hill-climbing local search pseudocode.

```
for k = 1 : K do
    Generate m random integer numbers idx_j ∈ {1,...,n}
    for j = 1 : m do
        Replace x_{k,idx_j} with all its possible values
        Evaluate the fitness function for the modified x_k
        Select the best modified particle
```

3.1.1.2 RNS-HPSO algorithm

Variable neighborhood search (VNS) algorithms, first introduced in [104], are meta-heuristics that do not follow a trajectory, but explore increasingly distant neighborhoods of the current incumbent solution and jump from this solution to a new one if, and only if, an improvement has been observed [105]. In the basic scheme, given a set of neighborhood structures, the VNS randomly generates a point x_{new} within the j-th neighborhood of the actual point x; afterwards, a local search modifies x_{new} and, only if x_{new} is better than x, it replaces the original point.

Standard and modified versions of the variable neighborhood search algorithms have been applied to solve different problems. In [106], a parallelized version of the VNS is proposed to enhance the speed of a meta-heuristic to solve the flexible job-shop scheduling problem. A VNS-based approach that can accept, with certain probability, deteriorating solutions is used in [107] to solve the dial-a-ride problem in a big city. VNS-based techniques have been applied also to the routing problem [108–110].

The random neighborhood search (RNS), derived from the reduced variable neighborhood search (RVNS) [111], was developed to solve the autonomous tug trajectory assignment problem. The structure of the j-th neighborhood of the particle x is defined as follows:

$$\mathcal{N}_j = \{x_{new} \mid \big(x_j - \Delta\big) \leq x_{j\ new} \leq \big(x_j + \Delta\big)\} \tag{3.2}$$

where x_j is the j-th element of the particle and Δ is the maximum allowed displacement from the actual value of x_j. In order to avoid the particles getting trapped in local optima, the value Δ is increased with a trend characterized by inverse proportionality with respect to the distance between x and $gbest$, as stated in Eq. 3.3:

$$\Delta = \max\left(\Delta_{min},\ \Delta_{max} \cdot \frac{eq_{el}}{n}\right) \tag{3.3}$$

with eq_{el} being the number of the elements of x that are equal to the corresponding elements of $gbest$; Δ_{min} and Δ_{max} are the minimum and maximum displacements that are peculiar to each element. The pseudocode of the RNS is described in Algorithm 5. In the proposed approach, the neighborhoods are analyzed in random order.

Algorithm 5 Random neighborhood search pseudocode.

```
for k = 1 : K do
    Generate m random integer numbers idx_j ∈ {1, ..., n}
    Compute the allowed displacement Δ for all the elements of x_k
    for j = 1 : m do
        Replace x_{k,idx_j} with a random value within the neighborhood N_{idx_j}
        Evaluate the fitness function for the modified x_k
        If the modified x_k has higher fitness, replace the original particle
```

3.1.1.3 Variable fixing algorithm

The particle swarm optimization uses the particle history to define the value of *pbest* (and accordingly of *gbest*); however, also the particle trajectory along the iterations could be exploited. In this work, a new solution to improve the rate of convergence and to reduce the number of fitness evaluations of the HPSO, by exploiting the particle trajectory, is proposed: the variable fixing algorithm (VarFix). This goal is achieved by setting the particle elements to the value characterized by higher probability of being part of the optimal solution. This feature leads the HPSO to focus on smaller parts of each particle, finding the optimal solution in fewer iterations. The VarFix can be applied to all the implementations of the particle swarm optimization, and potentially it can be adapted to other swarm algorithms and to the evolutionary algorithms. The number of elements m_{fix} to be modified by the hybridizing local search algorithm, after the VarFix activation, is given by Eq. 3.4

$$m_{fix} = \min(m, \quad n - \#fixed) \tag{3.4}$$

where the value m is chosen after a parameter tuning and $\#fixed$ is the number of elements that have been fixed for each specific particle.

In order to define if an element should be fixed, the evolution of different agents can be considered: the particle, the *pbest* or the *gbest* histories. If the particle history were considered, the variability of the position and fitness might prevent the algorithm to define with good accuracy the most promising value for a specific particle element. If the elements were fixed based on the *gbest* history, the algorithm could lead the particles to assume the same position in the search space; this is an undesired behavior, as the ability of the algorithm to escape local optima would be affected, due to the concentration of the particles in one specific area of the search space. Thus, the best approach is the one that uses the *pbest* as a discriminant for the fixing algorithm (Algorithm 6); indeed, the probability of defining with better accuracy the value to be set for each element is higher than the other approaches, as the *gbest* is always part of the *pbest* set.

Algorithm 6 Variable fixing algorithm pseudocode.

```
1: for k = 1 : K do
2:     if Iter ≥ Threshold then
3:         Evaluate pbest_k history
4:         for j = 1 : n do
5:             If pbest_{k,j} has been constant in the last α · Iter iterations,
                 fix pbest_{k,j}
6:             If pbest_{k,j} is fixed and Count ≥ rel_thold, release pbest_{k,j}
```

In order to be more accurate, the VarFix needs to collect information on the particle history. Therefore, the $Threshold$ constant in Algorithm 6 was defined as a minimum number of iterations up to which the VarFix only collects data. After the activation, the algorithm verifies, for each element in each particle, whether the value contained in the actual *pbest* has remained constant for a certain number of iterations, defined as a percentage of the actual iteration number $\alpha \cdot Iter$. If this condition is satisfied, the algorithm fixes the particle element. To avert the possibility of being

trapped in local minima, the VarFix releases the fixed elements after a certain number of iterations characterized by no improvements in the *gbest* solution (rel_{thold}).

A proper tuning of the algorithm parameters is required to ensure a trade-off between the necessity of improving the rate of convergence and the one of avoiding local optima, which would require to postpone the fixing. A parameter tuning analysis will be presented later in this section.

3.1.1.4 Convergence proof

The convergence of intelligent optimization algorithms is a crucial problem. Different approaches have been proposed in the literature to prove the convergence of meta-heuristics and their extension to stochastic combinatorial optimization, where convergence is a relevant issue [112–116].

An approach based on the theory of Markov chains is used to prove the convergence of the proposed hybrid particle swarm optimization [117, 118]. First, the convergence of the hybrid particle swarm optimization will be proved; then, a theorem for the convergence of the HPSO, with variable fixing, will be proposed.

Definition 3.1 (Markovian property)**.** *A stochastic process X_t is said to have Markovian property if the transition from one state to another depends only on the actual state of the process:*

$$\mathscr{P}\left(X_{t+1} = j \mid X_0 = i_0, X_1 = i_1, \ldots, X_t = i\right) = \mathscr{P}\left(X_{t+1} = j \mid X_t = i\right) \tag{3.5}$$

for $t = 0, 1, 2, \ldots$

Definition 3.2 (Homogeneous Markov chain)**.** *A stochastic process X_t is a Markov chain if it has the Markovian property. The conditional probability*

$$p_{ij}(t) = \mathscr{P}\left(X_{t+1} = j \mid X_t = i\right) \tag{3.6}$$

is called transition probability. A Markov chain is homogeneous if the conditional probability law $p_{ij}(t)$ does not depend on t.

Definition 3.3 (Communicating states)**.** *Given two states i and j, these states communicate if the following holds:*

$$\mathscr{P}\left(X_{t_1+k_1} = j \mid X_{t_1} = i\right) > 0 \ \wedge \ \mathscr{P}\left(X_{t_2+k_2} = i \mid X_{t_2} = j\right) > 0, \quad k_1, k_2 \geq 0 \tag{3.7}$$

Definition 3.4 (Class)**.** *A class is a collection of states that communicate with each other. A Markov chain is irreducible if there is only one class, thereby all the states communicate.*

Definition 3.5 (Absorbing state)**.** *A state i is said to be an absorbing state if, once entered in this state, the process will never leave it:*

$$\mathscr{P}\left(X_{t+1} = i \mid X_t = i\right) = 1 \tag{3.8}$$

Theorem 3.1. *The proposed hybrid particle swarm optimization applied to the set of finite states $\mathscr{S}$ is a homogeneous Markov chain.*

Proof. At each iteration t of the PSO, the probability $\mathscr{P}_{i,j}$ of moving from one state i to a state j depends only on the previous iteration $t-1$ and it is defined by an evolution law that does not change with time (Eq. 1.1). The final position of the particle, after the hybridizing algorithm, depends only on the position the particle had before the algorithm is applied. Thus, it follows that the HPSO algorithm with the set of finite states $\mathscr{S}$ belongs to the class of homogeneous Markov chain processes. ■

Theorem 3.2. *The HPSO is not an irreducible Markov chain.*

Proof. The implementation of the two proposed hybridizing heuristics prevents the HPSO from moving from a state (i) with a fitness value $f(i)$ to a state j with fitness $f(j) < f(i)$. If i_{min} is the state with global minimum fitness, we can write the following relation:

$$\mathscr{P}\left(X_{t+1} = i_{min} \mid X_t = i\right) = 0 \qquad \forall i \in \mathscr{S}, \quad \forall t \tag{3.9}$$

This implies that there exists at least one class composed of the sole element i_{min}, which is in contrast to the definition of an irreducible Markov chain. ■

Theorem 3.2 is fundamental as it implies that not all the states communicate; thus, the convergence cannot be directly proved. Before verifying the convergence

of the HPSO algorithm, two elements must be defined: the state *neighborhood* for the hybridizations, and the *attraction state*.

Definition 3.6 (Neighborhood)**.** *The state neighborhood of the state i is the set $\mathcal{N}_i$ of states visited by the hybridization algorithm while mutating the m randomly chosen elements of the particle in state i at iteration t.*

Definition 3.7 (Attraction state)**.** *Given a generic neighborhood $\mathcal{N}_i$, the state $j \in \mathcal{N}_i$ is an attraction state in $\mathcal{N}_i$ for the hybridizations if:*

$$f(j) \geq f(l) \qquad \forall l \in \mathcal{N}_i \setminus \{j\} \tag{3.10}$$

Thus, the HPSO moves from an *attraction state* to another. From Definition 3.7, it is possible to infer the following Theorem.

Theorem 3.3. *The global optimum state i^* is an attraction state for the neighborhood $\mathcal{N}$ if $i^* \in \mathcal{N}$.*

Proof. Theorem 3.3 can be proved by assuming that there exists a *neighborhood* $\mathcal{N}$ such that $i^* \in \mathcal{N}$, with an *attraction state* $i \neq i^*$; this would imply that $f(i) \geq f(i^*)$. By definition, i^* is the global optimum state, thereby it follows that $i \equiv i^*$, which contrasts with the initial assumption. ■

To prove the convergence of the algorithm, its ability of escaping local optima must be assessed; this is equivalent to proving that none of the *attraction states* is an absorbing state and thereby that there is no absorbing states in $\mathcal{S} \setminus \{i^*\}$.

Theorem 3.4. *All the attraction states $i \neq i^*$ are not absorbing states:*

$$\mathcal{P}(X_{t+1} = i \mid X_t = i) < 1 \qquad \forall i \in \mathcal{S} \setminus \{i^*\}, \quad \forall t \tag{3.11}$$

Proof. Consider a particle in state i at the beginning of iteration t, which was an *attraction state* at the iteration $t-1$. Since the inertia term of the particle velocity never goes to 0, the particle will move to a state $j \neq i$ even if i corresponds to *gbest* (that is also *pbest* for that particle). The search direction of the hybridizing algorithms and the *neighborhood* $\mathcal{N}_j$ are random; thereby it is possible to state the following:

$$\mathscr{P}\left(i \in \mathscr{N}_j\right) < 1 \Longrightarrow P\left(X_{t+1} = i \mid X_t = i\right) < 1 \tag{3.12}$$

Therefore, i is not an absorbing state.

Furthermore, even if $i \in \mathscr{N}_j$ and $i \neq i^*$, the stochastic evolution of the algorithm could lead with non-zero probability to a *neighborhood* $\mathscr{N}_j$ such that $\exists l \in \mathscr{N}_j \mid f(l) \geq f(i)$; thus, l would be the *attraction state* at iteration t. ■

It is therefore possible to prove the following Corollary.

Corollary 3.4.1. *Given Theorem 3.4 and the transition law of the HPSO, the following holds:*

$$\mathscr{P}\left(X_{t+k} \in \mathscr{N} \mid X_t \in \mathscr{M}\right) > 0 \qquad \forall \mathscr{N}, \mathscr{M} \subset \mathscr{N}_{\mathscr{S}}, \quad k \geq 1 \tag{3.13}$$

where $\mathscr{N}_{\mathscr{S}}$ is the collection of neighborhoods identified in the set of states $\mathscr{S}$.

Proof. Since all the *attraction states* are not absorbing points, the algorithm transits from an *attraction state* to another without restrictions and, thus, from a given *neighborhood* to another. ■

Based on Corollary 3.4.1, it is possible to think of all *neighborhoods* as being connected in a Markov chain sense; this implies that the HPSO algorithm can be seen as irreducible from a *neighborhood* point of view.

This characteristic is important as it forms the basis for the proof that the algorithm converges with probability 1.

Theorem 3.5. *The proposed hybrid particle swarm optimization algorithm asymptotically converges to the global optimum when time is endless.*

$$\mathscr{P}\left(X_{t+k} = i^* \mid X_t = i\right) = 1 \qquad \forall i \in \mathscr{S}, \quad k \geq 1 \tag{3.14}$$

Proof. Consider the time as endless; from Corollary 3.4.1 it directly descends that starting from a random state i at the initial iteration t_0, it exists a value $k \geq 1$ such that at iteration $t_0 + k$ the particle can move to a neighbor $\mathscr{N}$ such that $i^* \in \mathscr{N}$. Thus, for Theorem 3.3 the particle will converge to the global optimum defined by the state i^*. ■

Now it is possible to demonstrate that the global asymptotic convergence holds also when the variable fixing algorithm is added to the HPSO.

Theorem 3.6. *The HPSO with variable fixing asymptotically converges to the global optimum if rel_{thold} is finite.*

$$rel_{thold} \in \mathbb{Z}^{+} \Longrightarrow \mathscr{P}\left(X_{t+k} = i^{*} \mid X_{t} = i\right) = 1 \qquad \forall i \in \mathscr{S}, \quad k \geq 1 \tag{3.15}$$

Proof. The variable fixing algorithm reduces the search space of the HPSO to the set $\mathscr{R} \subset \mathscr{S}$ with $\mathscr{P}\left(i^{*} \in \mathscr{R}\right) < 1$. If $i^{*} \notin \mathscr{R}$ and $rel_{thold} = \infty$, the HPSO is not able to reach the global optimum. If rel_{thold} is finite, when the number of iterations in which the global best solution so far does not improves is equal to rel_{thold}, all the fixed elements are released and all the considerations done for the HPSO hold. Thus, the asymptotic convergence for the presented HPSO with variable fixing is proven. ■

3.1.1.5 Hybrid PSO for the trajectory assignment and departure sequencing problems

For the trajectory assignment application, the particle structure was defined as in Fig. 3.2. For each mission:

- $3 \times NF$ elements (in blue) represent the path index assigned with each flight, which is chosen among all the possible paths for that mission;
- $3 \times NF$ elements (in red) contain the speeds associated with each path;
- NF elements (in green) represent the PB_{buff} values ($PB_{buff} = 0$ for arrivals);
- $2 \times NF$ elements (in black) contain the *buff* variables.

$j = \{1,2,3\}$ is the index associated to each trajectory phase described in Sec. 2.2, for a total particle size of $9 \times NF$.

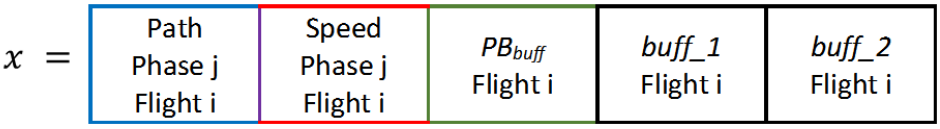


Fig. 3.2 Particle scheme for the trajectory assignment problem.

The fitness value of each particle is composed of two terms: the first one considers the total energy consumed by the tractors to perform the schedule represented by the particle (Eq. 2.21); the second term is a penalty function defined as the number of constraint violations (Eq. 2.22-2.26). Given that the PSO maximizes the fitness f, the latter is computed as the reciprocal of the linear combination of the two terms (Eq. 3.16).

$$f_k = \frac{1}{k_1 \cdot cost_k + k_2 \cdot penalty_k} \tag{3.16}$$

The weights k_1 and k_2 were chosen to give more importance to the penalty term of the fitness, thereby ensuring that the produced schedule is conflict-free; in the present work, the assigned values are: $k_1 = 1$ and $k_2 = 100$, in order to set the penalty term of the fitness at least one order of magnitude bigger than the energy consumption one. The value of k_2 can be tuned for different airports by considering the worst case cost for an average flight schedule in the specific airport, namely the cost resulting by the assignment of V_{max} to each phase of each tractor mission.

The trajectory assignment problem formulation presented in the previous chapter considers discrete and bounded variables; thus, some modifications are required to the HPSO to be suitable for this kind of problems. The maximum particle velocity value max_v_j was defined, for each element j, as reported in Eq. 3.17, where lb and ub represent the lower and upper bounds of each particle element.

$$max\_v_j = \frac{lb_j + ub_j}{2} - lb_j \tag{3.17}$$

If the velocity $v_{k,j}$ (Eq. 1.1) leads the element j of the particle k outside the bounds, the element position $x_{k,j}$ (Eq. 1.2) is set to the correspondent bound value (Eq. 3.18).

$$x_{k,j} = \begin{cases} ub_j & \text{if } (x_{k,j} + v_{k,j}) > ub_j \\ lb_j & \text{if } (x_{k,j} + v_{k,j}) < lb_j \\ x_{k,j} + v_{k,j} & \text{otherwise} \end{cases} \tag{3.18}$$

3.1.1.6 Stopping condition

The stopping criteria for both The HC-HPSO and the RNS-HPSO is composed of three terms: the maximum number of iterations is set to $maxIter = 300$; a condition is added that stops the algorithm if the *gbest* solution does not improve for a predefined number of iterations, set to the value $maxCount = 30$, in order to balance the performance and the computational time required to produce an optimal solution. The third term of the stopping condition is derived from the functional specifications, which state that the maximum allowed computational time is $15min$. If one of this stopping condition is met, the algorithm stops. If the algorithm meets the second criteria (*gbest* not improving), while the penalty value is not zero and the other two criteria are not met, the algorithm is allowed to continue running. This exception is granted to avoid solutions that generate conflicts among the tugs.

3.1.1.7 Parameter tuning

The algorithm parameters play a fundamental role in defining the performance of the algorithm; therefore, a proper tuning of all the parameters is required. In literature, several studies have been proposed on the parameter selection of the particle swarm optimization [119–122]. However, the presence of the hybridization adds the parameter m; therefore, a parameter tuning for the HC-HPSO and the RNS-HPSO was carried out on five benchmark problems. For each benchmark function, with problem dimension $n = 20$, $Nr = 50$ runs were performed for all the combinations of the set of parameters reported in Table 3.1. The coefficients of the particle velocity were set, following the studies reported in literature, to the values $c_1 = 2$, $c_2 = 2$ and W linearly decreasing with the iterations from 0.9 to 0.4.

Table 3.1 Hybrid particle swarm optimization algorithm parameter sets. Benchmark problems.

Parameter	**Values**			
K	$n/4$	$n/2$	$3 \cdot n/4$	n
m	$n/4$	$n/2$	$3 \cdot n/4$	n

The selected benchmark problems to tune the parameters are reported in Table 3.2 [123]; the variable ranges and the optimum solution are also reported. In this

work, the two hybrid particle swarm optimization algorithms were developed to solve discrete variable problems; therefore, the benchmark function variables were considered as discrete, with step $\Delta x = 0.01$.

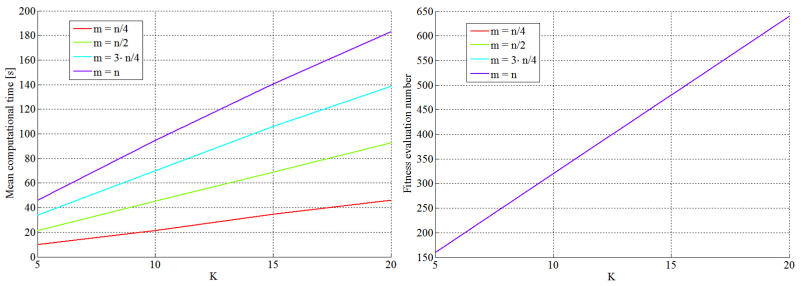
Table 3.2 Benchmark problems.

Function	$f(\bar{x})$	Variable domain	Optimum
Ackley's	$f_1(\bar{x}) = 20 + e - 20 \cdot \exp\left(-0.2 \cdot \sqrt{\frac{1}{n} \cdot \sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n} \cdot \sum_{i=1}^{n} \cos(2 \cdot \pi \cdot n)\right)$	$[-30,30]^D$	$f_1(\bar{0}) = 0$
Rastrigin's	$f_2(\bar{x}) = \sum_{i=1}^{n} \left(x_i^2 - 10 \cdot \cos(2 \cdot \pi \cdot x_i) + 10\right)$	$[-5.12,5.12]^D$	$f_2(\bar{0}) = 0$
Schwefel 1.2	$f_3(\bar{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_i\right)^2$	$[-65.53,65.53]^D$	$f_3(\bar{0}) = 0$
Sphere	$f_4(\bar{x}) = \sum_{i=1}^{n} x_i^2$	$[-5.12,5.12]^D$	$f_4(\bar{0}) = 0$
Rosenbrock's	$f_5(\bar{x}) = \sum_{i=1}^{n-1} \left(100 \cdot \left(x_{i+1} - x_i^2\right)^2 + (x_i - 1)^2\right)$	$[-30,30]^D$	$f_5(\bar{1}) = 0$

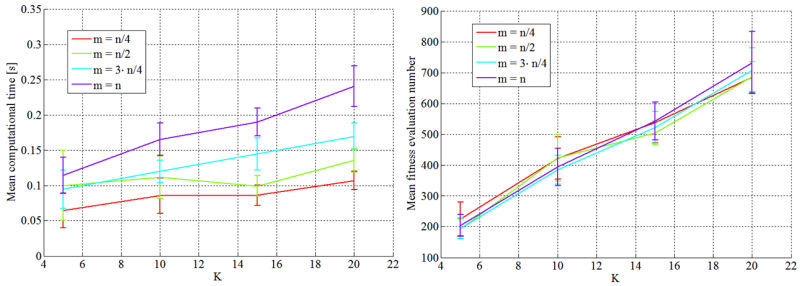
The mean value μ_{err} and standard deviation σ_{err} of the error on the fitness function value, showed in Table 3.3, were used as first comparison metric. Figures 3.3-3.7 illustrate the mean computational time and the number of function evaluations required to accomplish the optimization for each test case. The algorithm was implemented in C language, parallelized by means of the OpenMP®[124] library and executed on a Ubuntu 16.04 Pro platform supported by an Intel Core i7-4810MQ CPU and 8 GB RAM.

As reported in Table 3.3, the HC-HPSO always reaches the global optimum when applied to the first four benchmark problems; conversely, when the Rosenbrock's function is solved, the HC-HPSO always reaches a local optimum with mean error $\mu_{err} = 19.00$. The RNS-HPSO never finds the optimum for the proposed benchmark problems; however, conversely to what happens for the HC-HPSO, for the RNS-HPSO an improvement can be noticed in the results as the two parameters assume higher values. According to Figs.3.3-3.7, as expected, both the computational time and the fitness function evaluations are proportional to the number of particles in the swarm and increase with m.

To thoroughly analyze the dependence of the effectiveness and efficiency of the two algorithms on the two parameters K and m, two test cases for the trajectory assignment problems were implemented within the Turin airport. The first test case consists of $NF = 8$ flights, whereas the second one is characterized by a

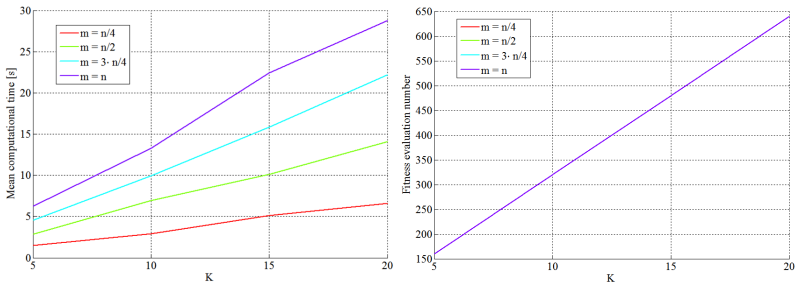


(a) HC-HPSO

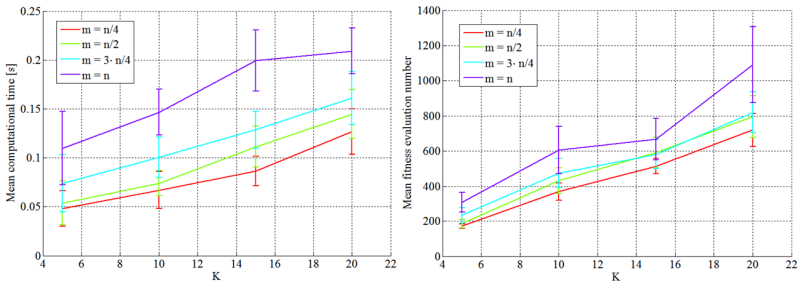


(b) RNS-HPSO

Fig. 3.3 Ackley's function. Mean computational time and number of function evaluations as a function of K and m



(a) HC-HPSO



(b) RNS-HPSO

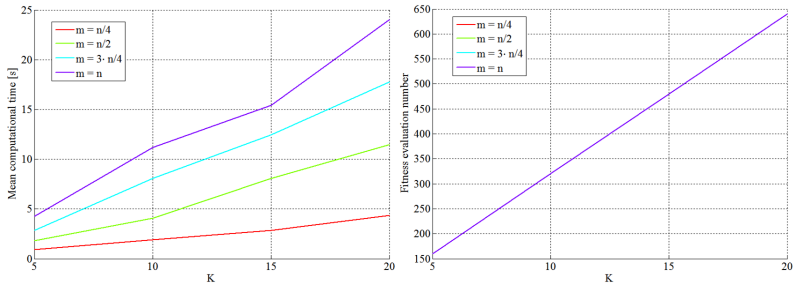
Fig. 3.4 Rastrigin's function. Mean computational time and number of function evaluations as a function of K and m

Table 3.3 Benchmark problems. Error mean and standard deviation. **The sign − indicates errors** $> 10^5$

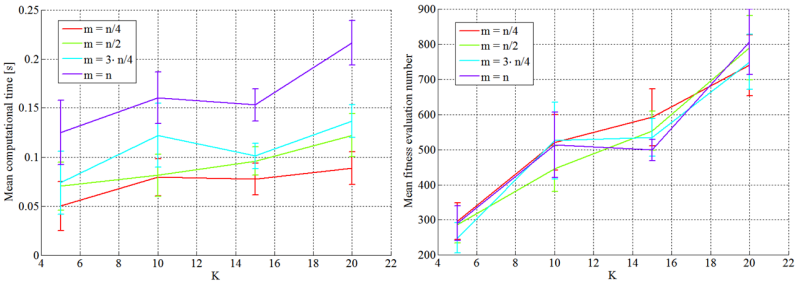
Algorithm	Parameter set K	m	f_1	f_2	f_3	f_4	f_5
HC-HPSO	$n/4$	$3/4$	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	$19.00 + 0.00$
		$n/2$	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	$19.00 + 0.00$
		$3 \cdot n/4$	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	$19.00 + 0.00$
		n	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	$19.00 + 0.00$
	$n/2$	$n/4$	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	19.00 ± 0.00
		$n/2$	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	19.00 ± 0.00
		$3 \cdot n/4$	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	19.00 ± 0.00
		n	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	19.00 ± 0.00
	$3 \cdot n/4$	$n/4$	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	19.00 ± 0.00
		$n/2$	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	19.00 ± 0.00
		$3 \cdot n/4$	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	19.00 ± 0.00
		n	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	19.00 ± 0.00
	n	$n/4$	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	19.00 ± 0.00
		$n/2$	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	19.00 ± 0.00
		$3 \cdot n/4$	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	19.00 ± 0.00
		n	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	19.00 ± 0.00
RNS-HPSO	$n/4$	$n/4$	19.63 ± 0.45	257.98 ± 25.33	9426.43 ± 1697.80	0.13 ± 0.09	–
		$n/2$	19.24 ± 0.45	233.56 ± 19.50	9224.67 ± 2156.35	0.05 ± 0.05	–
		$3 \cdot n/4$	18.93 ± 0.70	249.18 ± 19.71	8519.80 ± 1988.07	0.10 ± 0.54	–
		n	18.74 ± 0.51	259.28 ± 22.89	9360.50 ± 2310.07	0.17 ± 0.83	–
	$n/2$	$n/4$	19.46 ± 0.51	245.19 ± 13.85	7107.23 ± 1654.94	0.12 ± 0.08	–
		$n/2$	19.21 ± 0.37	240.78 ± 12.68	7090.52 ± 1628.88	0.09 ± 0.43	–
		$3 \cdot n/4$	18.79 ± 0.45	225.40 ± 9.87	6773.46 ± 1641.09	0.05 ± 0.19	–
		n	18.16 ± 0.83	243.02 ± 17.93	7078.93 ± 1911.32	0.57 ± 1.98	–
	$3 \cdot n/4$	$n/4$	19.01 ± 0.42	233.61 ± 18.59	6918.56 ± 1312.91	0.14 ± 0.12	–
		$n/2$	18.64 ± 0.65	226.91 ± 20.16	6040.91 ± 989.84	0.08 ± 0.15	–
		$3 \cdot n/4$	18.30 ± 0.95	232.06 ± 17.26	5444.42 ± 1081.55	0.10 ± 0.37	–
		n	17.50 ± 0.91	238.34 ± 17.33	5022.30 ± 896.64	0.09 ± 0.57	–
	n	$n/4$	18.80 ± 0.59	227.20 ± 18.21	5192.31 ± 1261.80	0.13 ± 0.13	–
		$n/2$	18.31 ± 0.60	230.72 ± 22.58	5742.44 ± 1303.71	0.24 ± 0.82	–
		$3 \cdot n/4$	18.16 ± 0.62	222.11 ± 26.55	6007.59 ± 1235.82	0.05 ± 0.23	–
		n	17.83 ± 0.74	211.71 ± 13.92	6212.91 ± 1236.86	0.38 ± 0.99	–

flight schedule containing $NF = 12$ flights. For each test case, $N_r = 50$ runs were performed for each combination of the parameter values reported in Table 3.4, with $n = 72$ and $n = 108$ for the first and second test cases respectively.

The results of the analysis are summarized in Table 3.5 in terms of mean and standard deviation of the energy consumed μ_{cost} and σ_{cost} and of the success rate $\%S$, namely the percentage of runs that led to conflict-free schedules for the tractors. It can be noticed that the improvement in the cost function, given by the incrementation of the two parameters, is almost negligible; in fact, it can be evaluated in 10^{-1} kWh

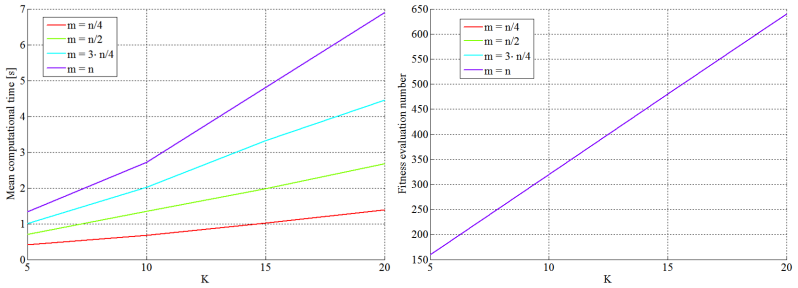


(a) HC-HPSO

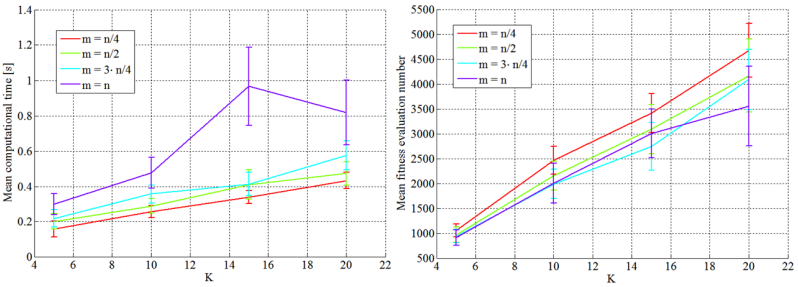


(b) RNS-HPSO

Fig. 3.5 Schwefel 1.2 function. Mean computational time and number of function evaluations as a function of K and m

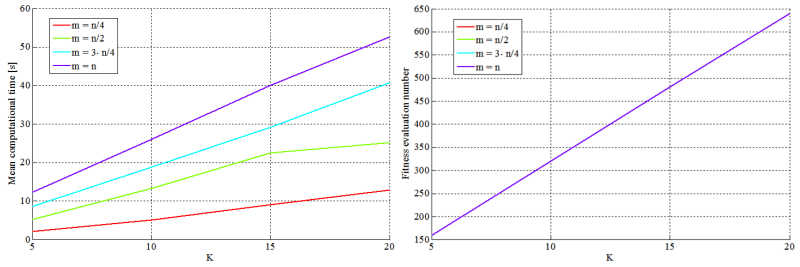


(a) HC-HPSO

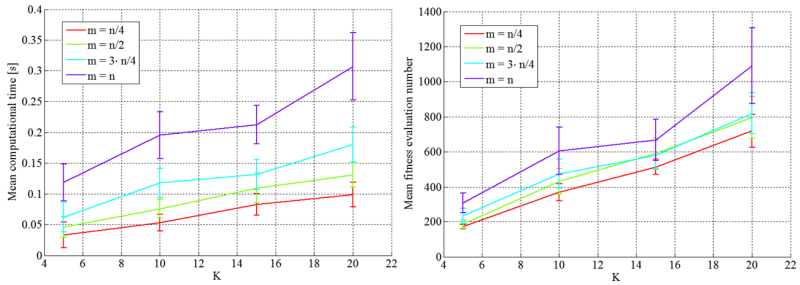


(b) RNS-HPSO

Fig. 3.6 Sphere function. Mean computational time and number of function evaluations as a function of K and m



(a) HC-HPSO



(b) RNS-HPSO

Fig. 3.7 Rosenbrock's function. Mean computational time and number of function evaluations as a function of K and m

Table 3.4 Hybrid particle swarm optimization algorithm parameter sets. Trajectory assignment problem.

Parameter	**Values**			
K	2	8	16	
m	$n/4$	$n/2$	$3 \cdot n/4$	n

for both the hybridizations of the particle swarm optimization. Differently from what was seen for the benchmark problems, when solving the trajectory assignment problem test cases, the performance of the RNS-HPSO are close to the one of the HC-HPSO. As far as the efficiency of the algorithms is concerned, Figs. 3.8-3.9 show the mean computational time and the mean number of fitness function evaluations for both algorithms. Also in this case, it is possible to identify a proportional trend of the computational time and function evaluation number with the two parameters.

Given the previous considerations, the selected values for the two parameters are $K = 8$ and $m = 3 \cdot n/4$, which grant good effectiveness, while requiring a reasonable computational time to solve the problem.

Table 3.5 Trajectory assignment problems. The results are presented in the form: $\mu_{cost} \pm \sigma_{cost}$ $(\%S)$.

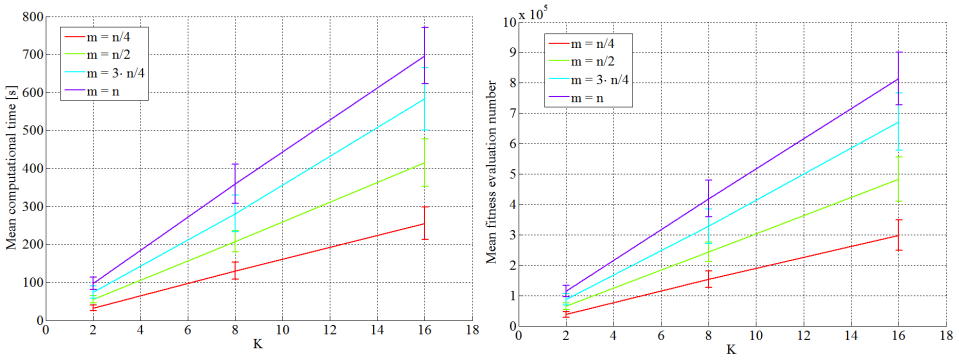
Algorithm	Parameter set K	m	Test case 1	Test case 2
HC-HPSO	2	$n/4$	42.96 ± 1.49 (100)	59.92 ± 2.63 (58)
		$n/2$	42.93 ± 0.68 (100)	60.23 ± 3.02 (98)
		$3 \cdot n/4$	42.39 ± 1.23 (100)	59.41 ± 1.88 (100)
		n	42.30 ± 0.96 (100)	59.53 ± 1.44 (100)
	8	$n/4$	42.95 ± 0.73 (100)	62.24 ± 3.39 (88)
		$n/2$	42.58 ± 0.35 (100)	60.13 ± 1.86 (100)
		$3 \cdot n/4$	42.30 ± 1.16 (100)	59.64 ± 1.18 (100)
		n	42.39 ± 0.25 (100)	59.29 ± 1.67 (100)
	16	$n/4$	42.75 ± 0.50 (100)	62.16 ± 2.74 (100)
		$n/2$	42.19 ± 1.52 (100)	60.09 ± 1.97 (100)
		$3 \cdot n/4$	42.41 ± 0.84 (100)	59.62 ± 1.26 (100)
		n	42.28 ± 1.14 (100)	59.71 ± 1.36 (100)
RNS-HPSO	2	$n/4$	43.97 ± 1.50 (98)	62.09 ± 2.58 (42)
		$n/2$	43.60 ± 1.23 (100)	61.24 ± 2.37 (68)
		$3 \cdot n/4$	43.42 ± 0.72 (100)	62.07 ± 2.65 (96)
		n	43.32 ± 0.85 (100)	61.43 ± 2.25 (96)
	8	$n/4$	43.38 ± 0.81 (100)	63.07 ± 3.05 (74)
		$n/2$	43.39 ± 0.61 (100)	63.21 ± 3.35 (96)
		$3 \cdot n/4$	43.40 ± 0.52 (100)	62.78 ± 2.44 (100)
		n	43.15 ± 0.52 (100)	62.39 ± 2.58 (100)
	16	$n/4$	43.27 ± 0.73 (100)	63.53 ± 2.99 (90)
		$n/2$	43.24 ± 0.53 (100)	63.20 ± 2.41 (100)
		$3 \cdot n/4$	43.09 ± 0.40 (100)	62.37 ± 2.64 (100)
		n	43.14 ± 0.34 (100)	61.97 ± 2.48 (100)

In order to tune the parameters of the variable fixing algorithm, the same benchmark problems were used for the combinations of the parameter values reported in Table 3.6. The HPSO parameters were set to the values resulting from the previous analysis.

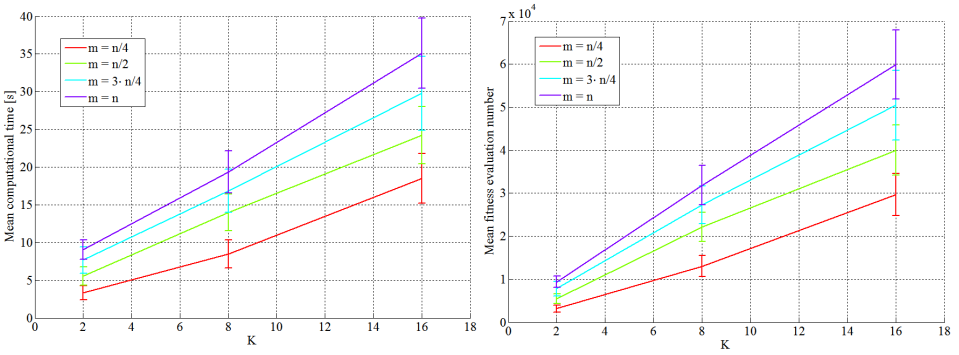
Table 3.6 Variable fixing algorithm parameter sets.

Parameter	Values		
$Threshold$	10	20	30
α	0.1	0.2	0.3
rel_{thold}	5	10	20

Table 3.7 shows the mean ad the standard deviation of the fitness error for both the HPSO with and without the variable fixing and for all the parameter sets. As it can be noticed, the introduction of the VarFix does not affect the error in the solution cost for the proposed problems; indeed, the best solution has always appeared before



(a) HC-HPSO



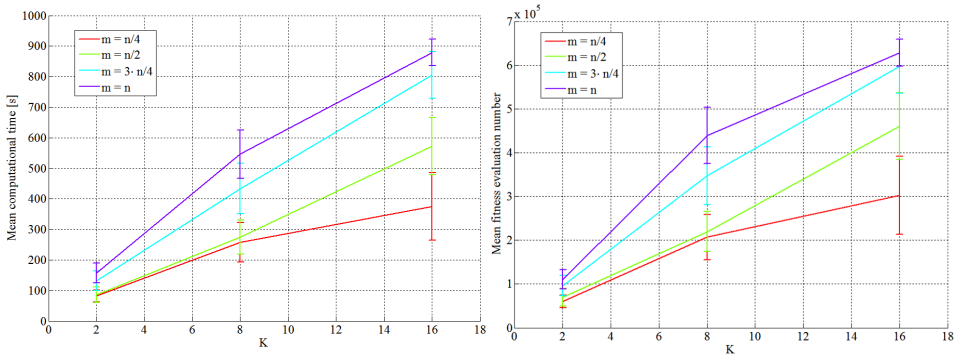
(b) RNS-HPSO

Fig. 3.8 LIMF test case 1. Mean computational time and number of function evaluations as a function of K and m

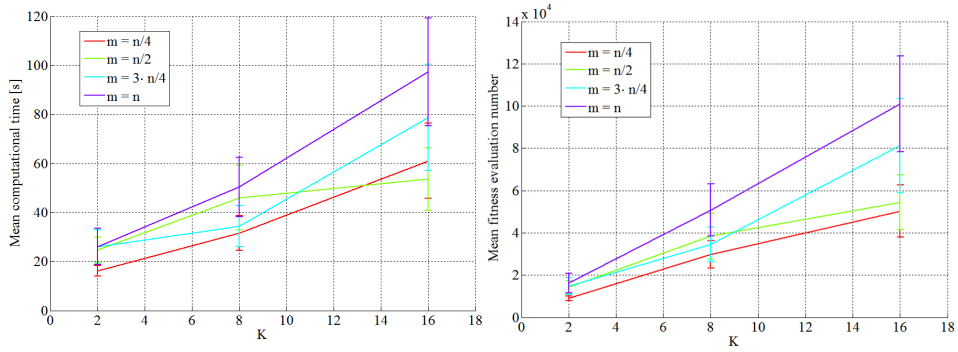
Table 3.7 Benchmark problems. Error mean and standard deviation.

Algorithm	**Parameter set**	$\mathbf{f_1 - f_4}$	$\mathbf{f_5}$
HPSO	-	$00.00E+00$	$19.00E+00$
HPSO w/ variable fixing	all	$00.00E+00$	$19.00E+00$

the variable fixing algorithm had started to actively operate. The results presented in Table 3.7 show also that the HPSO algorithm was not able to find the optimal solution for the Rosenbrock's function, but it always got stuck in the local minimum corresponding to the solution $f_5\left(\bar{0}\right)$. Figs. 3.10-3.14 demonstrate that, for the proposed problems, the variable fixing algorithm was always capable of reducing the computational time. However, the algorithm parameters have different effects on the required computational time: as both the parameters α and $Threshold$ increase, the computational time increases, whereas the rel_{thold} does not significantly affect the performance of the algorithm for $Threshold > 10$.



(a) HC-HPSO



(b) RNS-HPSO

Fig. 3.9 LIMF test case 2. Mean computational time and number of function evaluations as a function of K and m

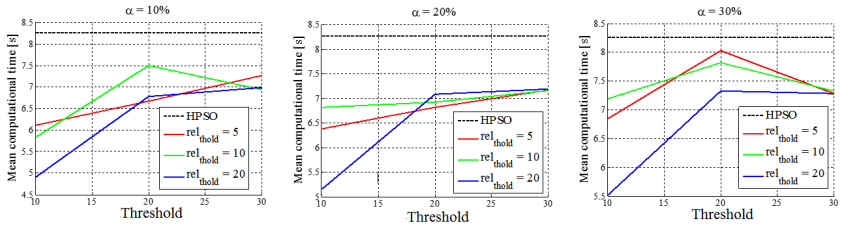


Fig. 3.10 Ackley's function computational time.

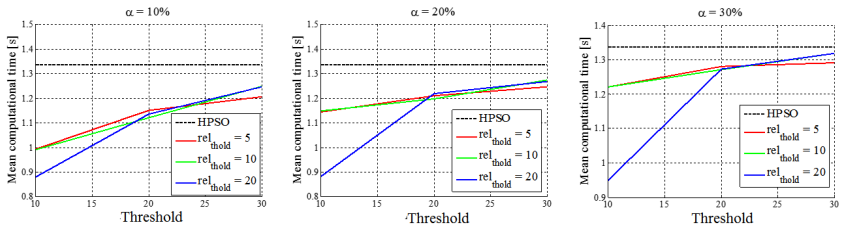


Fig. 3.11 Rastrigin's function computational time.

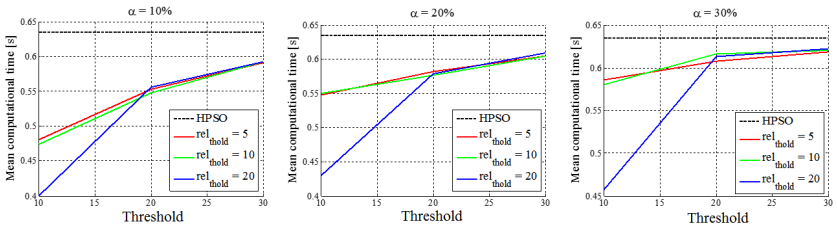


Fig. 3.12 Schwefel's function computational time.

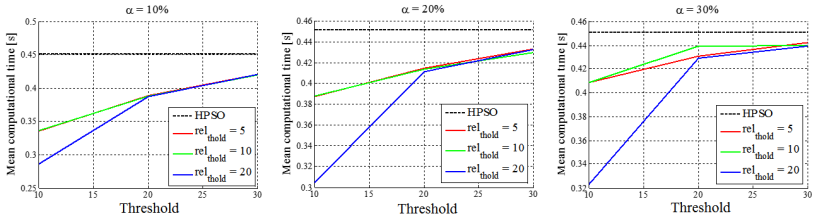


Fig. 3.13 Sphere's function computational time.

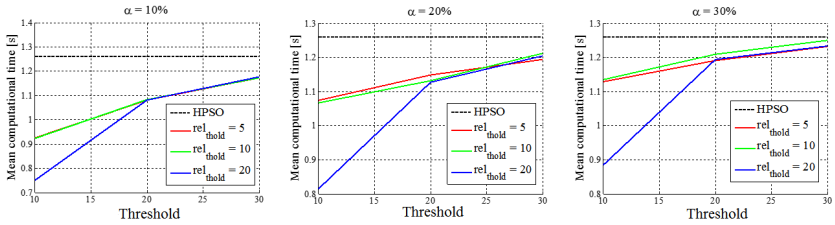
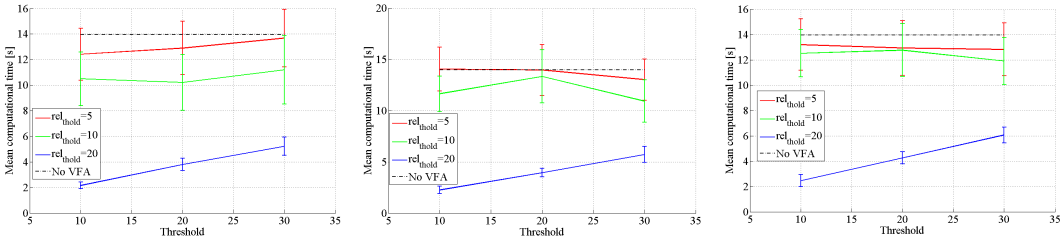


Fig. 3.14 Rosenbrock's function computational time.

The results of the benchmark problems do not provide any information on the effects of the variable fixing algorithm parameters on the best solution; however, they show that, as a general rule, the lower the value of the parameters, the lower the computational time required to find the solution. Further information can be gathered from the application of the variable fixing algorithm to the two test cases for the trajectory assignment problem previously proposed.

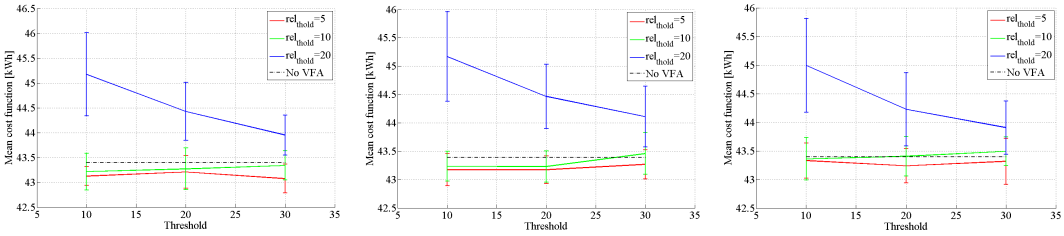
Figures 3.15-3.18 show the mean computational time and the mean cost for each combination of the parameters $Threshold$, rel_{thold} and α. As it can be noticed, the variable fixing algorithm was effective in decreasing the computational time required for almost all the parameter combinations. Besides, when $\alpha \leq 20\%$ and $rel_{thold} \leq 10$ were considered, the VarFix was capable also to lead the hybrid particle swarm optimization to solutions characterized by lower cost function values.

Considering the results of the parameter tuning analysis for the variable fixing algorithm, the set of values selected for the three parameters are: $Threshold = 20$, $rel_{thold} = 10$ and $\alpha = 10\%$.



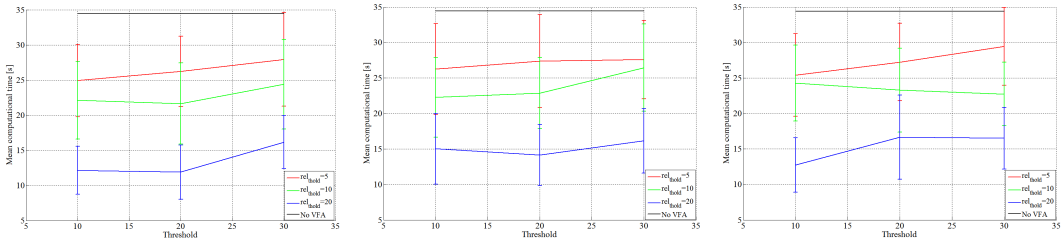
(a) $\alpha = 10\%$ (b) $\alpha = 20\%$ (c) $\alpha = 30\%$

Fig. 3.15 LIMF test case 1. Mean computational time as a function of $Threshold$, rel_{thold} and α



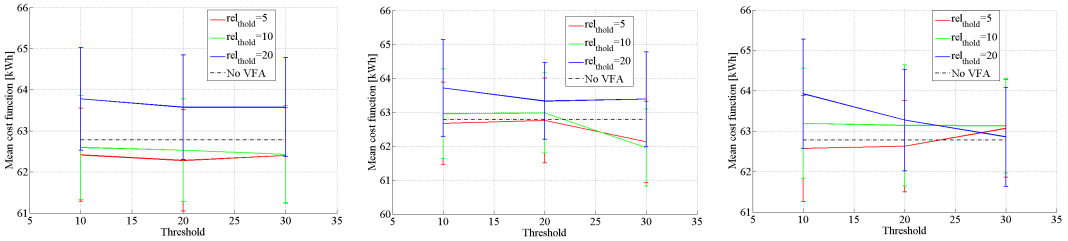
(a) $\alpha = 10\%$ (b) $\alpha = 20\%$ (c) $\alpha = 30\%$

Fig. 3.16 LIMF test case 1. Mean cost function value as a function of $Threshold$, rel_{thold} and α



(a) $\alpha = 10\%$ (b) $\alpha = 20\%$ (c) $\alpha = 30\%$

Fig. 3.17 LIMF test case 2. Mean computational time as a function of $Threshold$, rel_{thold} and α



(a) $\alpha = 10\%$ (b) $\alpha = 20\%$ (c) $\alpha = 30\%$

Fig. 3.18 LIMF test case 2. Mean cost function value as a function of $Threshold$, rel_{thold} and α

3.1.2 Tree search heuristic

The runway utilization can be modeled as a switch that is closed (value 1) when an airplane is landing or taking off and is open (value 0) when the runway is free. Given a flight schedule, the scheme of the runway occupation as a function of time, caused by the landing flights, can be depicted as in Fig. 3.19, where A_j represents the j-th arrival and S_i the i-th departure slot. Hence, the starting and ending time of the slots that can be used by departing airplanes can be computed.

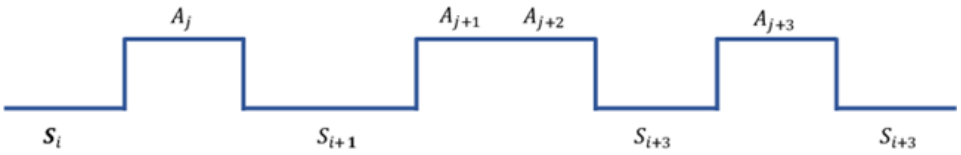


Fig. 3.19 Runway occupation scheme.

The earliest (ERT) and latest (LRT) time at runway for a departing aircraft can be computed by means of Eq. 3.19 and Eq. 3.20, where t_{PB} is the earliest pushback time and d_{s-r} is the distance between the aircraft stand and the runway entry. In the computation of ERT and LRT no speed ramp-up and ramp-down phases are considered.

$$ERT = t_{PB} + \frac{d_{s-r}}{V_{max}} \tag{3.19}$$

$$LRT = t_{PB} + \frac{d_{s-r}}{V_{min}} + MaxBuffer \tag{3.20}$$

Therefore, given the *ERT* and *LRT* of each departure, it is possible to enumerate all the departure sequences and collect them in a tree representation that contains in each branch a possible assignment of departures to slots (Fig. 3.20). However, the assignment of one aircraft to a specific slot can prevent the assignment of other airplanes to it, due to the runway separation requirements and the finite amplitude of the slots. This behavior can lead to the creation of branches that do not assign all the departing airplanes to a slot, thereby creating unfeasible solutions that were immediately pruned, as highlighted in red in Fig. 3.20, which represent a hypothetical case where the aircraft can depart only in the slots defined in Table 3.8. In this case, departure D_3 cannot be assigned in the sequences circled in red, which must be pruned.

Table 3.8 Hypothetical departure sequencing problem.

Departure	Possible slots
D_1	$[S_1]$
D_2	$[S_1, S_2]$
D_3	$[S_2, S_3]$
D_4	$[S_3]$

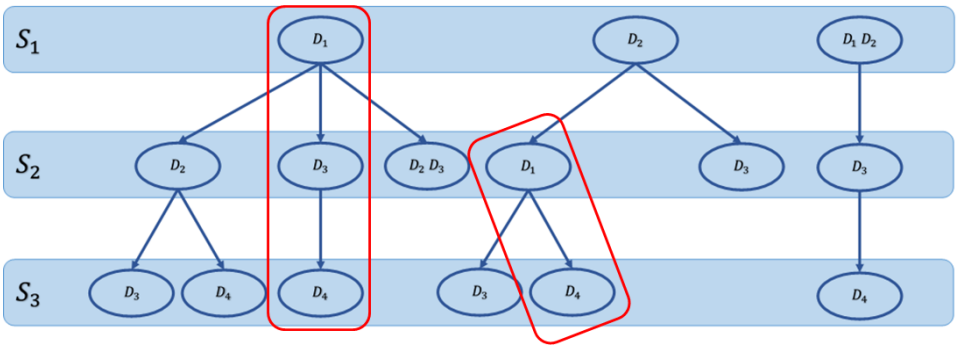


Fig. 3.20 Runway sequencing tree representation with unfeasible sequences highlighted.

The proposed tree search heuristic is based on a greedy algorithm with recovery feature that aims to find the best branch in the departure sequencing tree: the electric vehicle optimal scheduling (EVOS) heuristic. A cost value is computed for each branch and the one with the lowest cost is chosen as solution. The cost is estimated by running the optimization algorithm to define a (near-)optimal tractor schedule satisfying the slot assignment. Therefore, the constraints defined in Eq. 3.21 and Eq. 3.22 are added to both the discrete and the continuous time models presented in the previous chapter:

$$start_{slot_{asgd}}(i) \leq t_{RWY}(i) \qquad i \in \textbf{TOs} \tag{3.21}$$

$$t_{RWY}(i) + t_{TO}(i) \leq end_{slot_{asgd}}(i) \qquad i \in \textbf{TOs} \tag{3.22}$$

where the start and the end of the assigned slot for the i-th departure are respectively $start_{slot_{asgd}}(i)$ and $end_{slot_{asgd}}(i)$. The arrival time at the runway is $t_{RWY}(i)$, whereas $t_{TO}(i)$ is the duration of the take-off.

The additional constraints define a strict boundary in the range of taxiing speeds that can be allocated to each departure; in fact, the lowest and the highest speeds that can be used by the tractor to satisfy the slot assignment can be computed by means of Eq. 3.23 and Eq. 3.24 respectively.

$$V_{low} = \frac{d_{s-r}}{end_{slot_{asgd}} - Disc_{time} - t_{PB} - PB_{buff}} \tag{3.23}$$

$$V_{high} = \frac{d_{s-r}}{start_{slot_{asgd}} - Disc_{time} - t_{PB} - PB_{buff}} \tag{3.24}$$

The EVOS algorithm aims to find the optimal or near-optimal conflict-free schedule, minimizing the deviations from the lower bound schedule for the unconstrained problem (no conflict considered). In order to find the lower bound for the unconstrained problem, the consumption model presented in Section 2.3 must be considered. The energy consumption is proportional to the traveling speed V and the traveled distance d. Hence, the lower bound schedule is the one that assigns to each vehicle the shortest path and the speed that implies minimum energy consumption $V_{E_{min}}$. To a first approximation, considering only the constant speed segment of each mission phase, the minimum energy consumption speed can be computed starting from Eq. 2.12, which for simplicity is reported below:

$$\Delta E_{tot} = \frac{d}{\eta_{out} \cdot \eta_d \cdot 3600} \cdot \left(\frac{P_1 \cdot V^2 + P_2}{\eta_{EM}} + P_{aux} \cdot V^{-1} \right)$$

As mentioned above, the minimum energy consumption is obtained by traveling on the shortest path with length d_{min}; thereby it is possible to define a coefficient for the minimum energy $q_{min} = d_{min} / (\eta_{out} \cdot \eta_d \cdot 3600)$. The speed that corresponds to

the minimum can be then computed posing the derivative of the energy consumption equal to zero, as reported in Eq. 3.25. As it can be noticed, the minimum energy consumption speed depends only on the aerodynamic features of the tractor (or tractor and airplane), with constant of proportionality 8.7721, which comes from the data in Table 2.1.

$$\begin{aligned} &\frac{d}{dV}\Delta E_{tot} = 0 \quad \Longrightarrow \quad q_{min} \cdot \left(2 \cdot \frac{P_1}{\eta_{EM} \cdot V_{E_{min}}} - P_{aux} \cdot V_{E_{min}}^{-1}\right) = 0 \quad \Longrightarrow \\ &\Longrightarrow \quad V_{E_{min}} = \sqrt[3]{\frac{P_{aux} \cdot \eta_{EM}}{2 \cdot P_1}} = \sqrt[3]{\frac{P_{aux} \cdot \eta_{EM}}{2}} \cdot \sqrt[3]{P_1^{-1}} = 8.7721 \cdot \sqrt[3]{P_1^{-1}} \end{aligned} \tag{3.25}$$

The proposed greedy algorithm analyzes one mission at a time, following the flight schedule order; the shortest path and $V_{E_{min}}$ are assigned to the three phases of the considered mission. Conversely to the classical greedy algorithms, the proposed one has a recovery feature that allows it, if needed, to change choices that have been previously made. This feature is exploited in the situation where conflicts arise with the previous assigned trajectories; therefore, the speed of the involved vehicles and the buffer time are modified via a reduced variable neighborhood search (RVNS), to find a conflict-free solution that minimizes the total cost at this decision point. A graphical representation of how the algorithm works is reported in Fig. 3.21, where V_{opt1} and V_{opt2} are the optimal speed values, found for each flight after the first or second step of the algorithm respectively. The pseudocode of the optimization heuristic is presented in Algorithm 7.

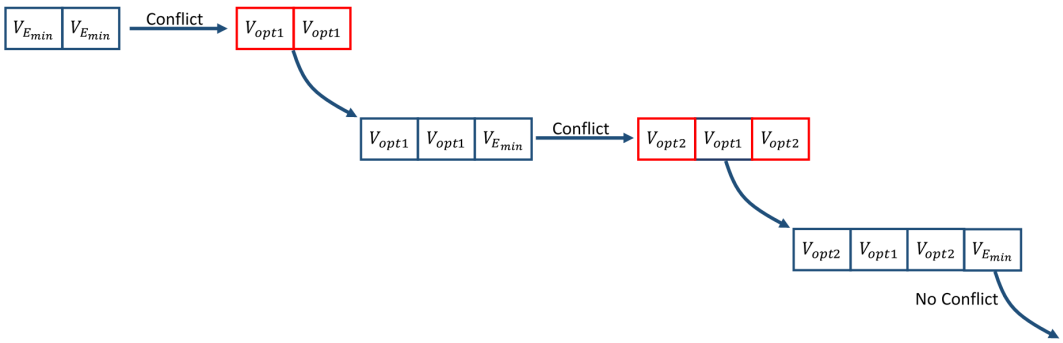


Fig. 3.21 Reasoning representation of the greedy EVOS algorithm.

The proposed RVNS analyzes the incumbent solution neighborhoods in random order (Algorithm 8): m random integers are generated in the set $\{1,\ldots,Ns\}$, with Ns length of the solution array s, which contains the three speeds and three buffer

Algorithm 7 Electric vehicle optimal scheduling tree search heuristic pseudocode.

```
Enumerate all the possible combinations of runway slot assignment
Build the feasible slot assignment tree
while (Stopping condition) = FALSE do
    for all branches do
        Set V=V_{E_min} for each phase of the first mission
        for all i ∈ {2,...,NF} do
            Set V=V_{E_min} for the phases of the i-th mission
            Compute the fitness after the addition of the i-th mission
            If any conflict or any constraint violation are detected, run the RVNS
```

time of the actual considered mission and of the Nc conflicting ones. Thereby, the array size is $Ns = 6 \cdot (Nc + 1)$. For the j-th generated index, the solution array is randomly changed within the j-th neighborhood (Eq. 3.26) until the associated fitness function stops growing. The RVNS is then repeated $maxIter_{RVNS}$ times, using the final solution found in the previous iteration as a tentative solution for the new iteration.

$$\mathcal{N}_j = \{s' \mid lb_{s_j} \leq x'_j \leq ub_{s_j}\} \tag{3.26}$$

Algorithm 8 Reduced variable neighborhood search pseudocode.

```
for i = 1 : MaxIter_RVNS do
    Generate m random integer numbers idx_j ∈ {1,...,Ns}
    for j = 1 : m do
        while fitness > fitness_old do
            Replace s_{idx_j} with a random value within the neighborhood N_{idx_j}
            Evaluate the fitness function for the modified s
            If the new s has higher fitness, replace the original solution vector
```

3.1.2.1 Stopping condition

At the end of each iteration, the algorithm checks if a feasible solution has been found, at least in one of the branches; if this condition is met, the algorithm stops. In the case that the previous condition is violated, the algorithm runs again for a maximum number of iterations $maxIter = 5$. Similarly to what has been done for the hybrid particle swarm optimization, a third constraint is defined to prevent the

algorithm overrunning the functional requirement of maximum computational time equal to 15 *min*.

3.1.2.2 Parameter tuning

The solution found by the tree search algorithm is influenced by the number of times the reduced variable neighborhood search algorithm is run for each conflict resolution $MaxIter_{RVNS}$ and by the number of elements m of the solution array, which are changed within the reduced variable neighborhood search. A parameter tuning was carried out on the same test cases within the "*Sandro Pertini*" Turin airport used for the parameter tuning of the HPSO. The set of parameters used for the tuning is presented in Table 3.9.

Table 3.9 Tree search algorithm parameter sets.

Parameter	**Values**				
$MaxIter_{RVNS}$	5	10	20	30	50
m	$Ns/2$	Ns			

For each parameter value combination, $Nr = 50$ runs per test case were executed; the mean μ_{cost} and standard deviation σ_{cost} of the cost function expressed in 2.21 and the success rate $\%S$ were used as comparison metric. Table 3.10 illustrates the results for each test case; Fig. 3.22 and Fig. 3.23 show the mean and standard deviation of the computational time and of the fitness evaluations.

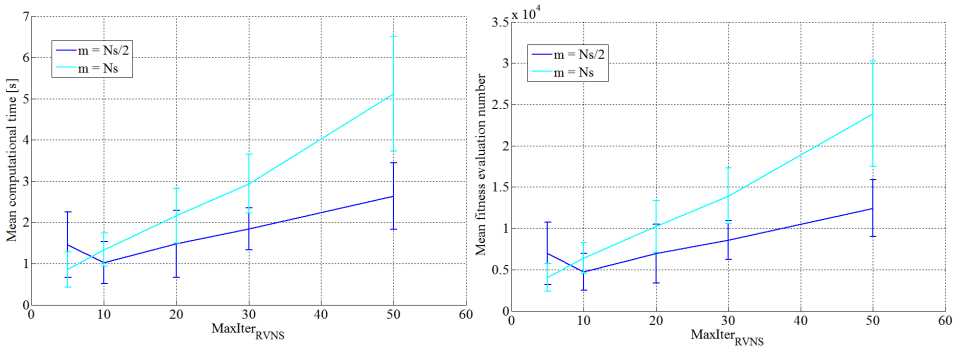


Fig. 3.22 LIMF test case 1. Mean computational time and number of function evaluations as a function of $MaxIter_{RVNS}$ and m

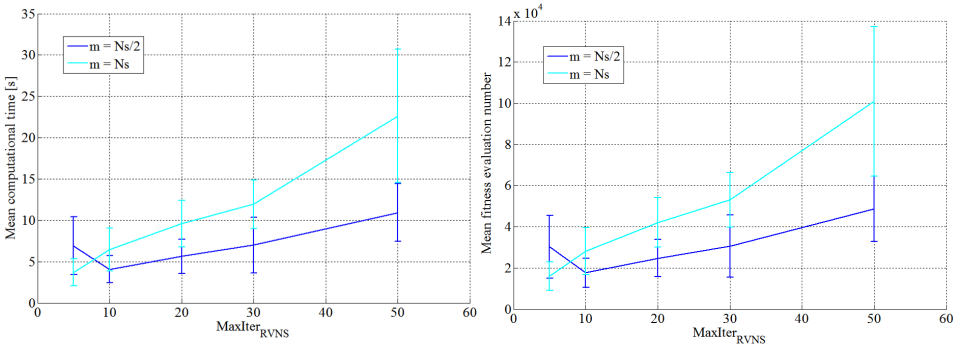


Fig. 3.23 LIMF test case 2. Mean computational time and number of function evaluations as a function of $MaxIter_{RVNS}$ and m

Table 3.10 Trajectory assignment problems. The results are presented in the form: $\mu_{cost} \pm \sigma_{cost}$ $(\%S)$.

Algorithm	Parameter set		Test case 1	Test case 2
	$\text{MaxIter}_{\text{RVNS}}$	m		
TS	5	$Ns/2$	60.16 ± 3.98 (100)	95.05 ± 6.86 (100)
		Ns	55.54 ± 5.31 (100)	88.99 ± 6.82 (100)
	10	$Ns/2$	57.33 ± 4.49 (100)	91.64 ± 8.78 (100)
		Ns	55.09 ± 2.91 (100)	85.47 ± 8.02 (100)
	20	$Ns/2$	54.25 ± 4.11 (100)	84.32 ± 6.88 (100)
		Ns	54.92 ± 3.11 (100)	80.10 ± 6.99 (100)
	30	$Ns/2$	55.71 ± 2.69 (100)	81.02 ± 5.91 (100)
		Ns	55.00 ± 2.62 (100)	81.04 ± 6.93 (100)
	50	$Ns/2$	54.26 ± 3.60 (100)	80.06 ± 5.32 (100)
		Ns	53.51 ± 3.89 (100)	78.23 ± 6.14 (100)

Based on the results of the simulations, the parameters were set to the values $MaxIter_{RVNS} = 30$ and $m = NS/2$.

3.2 Tractor dispatch algorithm

The tractor dispatch problem is a scheduling problem with consumable resources (tractors) that are used to perform actions (departures/arrivals); the resources can be restored (battery charging) during the process. The scheduling problem was

proved to be NP-hard, justifying the choice of approximate methods to solve it in a limited computational time. The hybrid particle swarm optimization algorithms presented to solve the trajectory assignment problem were used also to find the optimal assignment of the tractors to the missions. The particles for the considered problem assume the structure depicted in Fig. 3.24. The size of each particle is $n = NF$, with lower bound $lb_k = 1$ and upper bound $ub_k = NT$ for each element x_k, where NT is the fleet size.

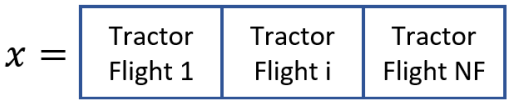


Fig. 3.24 Particle structure for the tractor dispatch algorithms.

In medium and big airports, more than one tractor depot can be defined, thereby establishing multiple fleets with size NT_i, with $i \in \{1,\ldots,Nd\}$ and Nd number of tractor fleets in a specific airport. In this work, disjoint depots were considered: a tractor that is initialized in a specific depot, can not move to another depot. The minimum number of tractors required in each fleet to perform the missions can be established running the aforementioned algorithms with increasingly lower fleet sizes.

Chapter 4

Simulation Results

To test the performance of the developed optimization algorithms and their reconfigurability, they were applied to a set of test case scenarios in different airports; both the optimization models presented in Chapter 2 were used. Three airports with different characteristics were considered: a single-runway low-traffic airport (i.e. the "*Sandro Pertini*" Turin airport), a two-runways medium-traffic airport with two separate terminals (i.e. the Milan Malpensa airport) and a large hub high-traffic airport (i.e. the Amsterdam Schiphol airport). For each airport, the model was developed using the GUI presented in Section 2.1.

4.1 Test case scenarios

This section gives an overview on how the three considered airports were modeled and introduces the flight schedules of the test case scenario and explains how they have been built.

4.1.1 Turin airport

The layout of the Turin airport consists of a main two-way taxiway and a one-way taxiway traveling along the main apron, as depicted in Fig. 4.1. Only one tractor depot area was defined for this airport; it has been located on the side of the terminal building (black rectangle area).

Fig. 4.1 Turin airport layout with indication of the taxiway direction and of the depot. Credits: Pictures 2015 DigitalGlobe, Map data ©2015 Google

Table 4.1 Airplane mechanical and aerodynamic characteristics. MTOW: maximum take off weight, MLW: maximum landing weight.

MTOW [kg]	MLW [kg]	Length [m]	Wingspan [m]	C_L	S $[m^2]$	Landing speed [m/s]
78000	66000	37.57	34.10	1.96	122.60	72.01

No data on aircraft stands and runway entries assignment for real flight schedules were available. Therefore, a flight schedule generator was designed taking into consideration the actual flight schedules of the Turin airport in different periods of the year, retrieved from the airport website. The average traffic during the peak hours (from 9.00 to 13.00) is 7 departures/arrivals per hour; the interval between two arrivals was randomly generated with uniform distribution between the values $40s$ and $900s$. The turnaround time for each flight was generated in the range $20-50min$; gates were randomly assigned, whereas the runway entries/exits were allocated to comply with the actual usage of the runway, which is mostly in configuration 36. The aircraft stand codes and locations and the runway entry codes were extracted from the airport information publication (AIP), reported in the ENAV website [125]. All the flights were considered to be operated by aircraft with the characteristics reported in Table 4.1.

The touch down (TD) time and pushback (PB) time were then defined as follows:

$$t_{TD,i} = t_{TD,i-1} + U(40,900) \qquad \forall i = 1,\ldots,NF \tag{4.1}$$

$$t_{PB,i} = t_{TD,i} + U(1200,3000) \qquad \forall i = 1,\ldots,NF \tag{4.2}$$

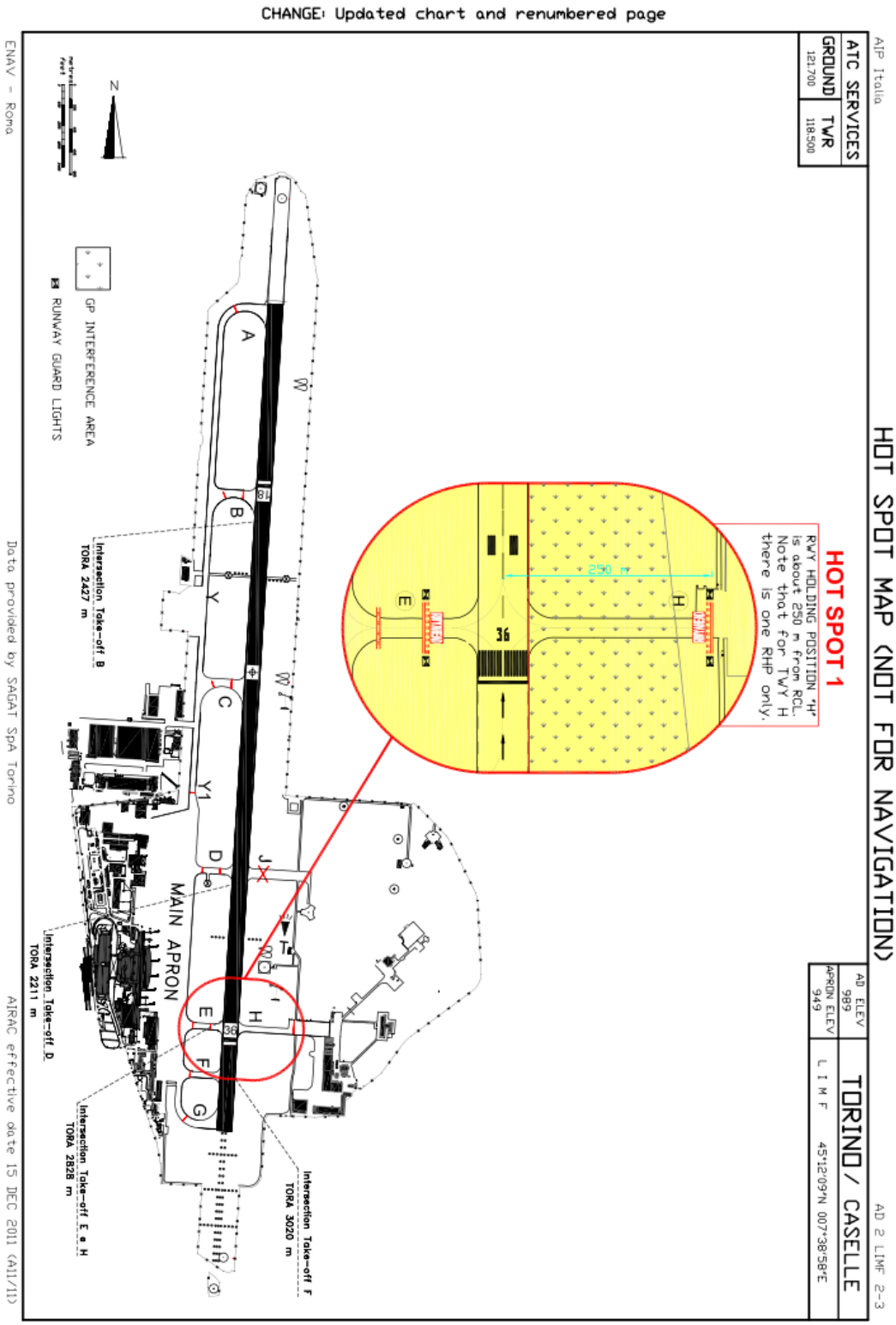


Fig. 4.2 Turin airport layout with indication of the runway entry labels [125].

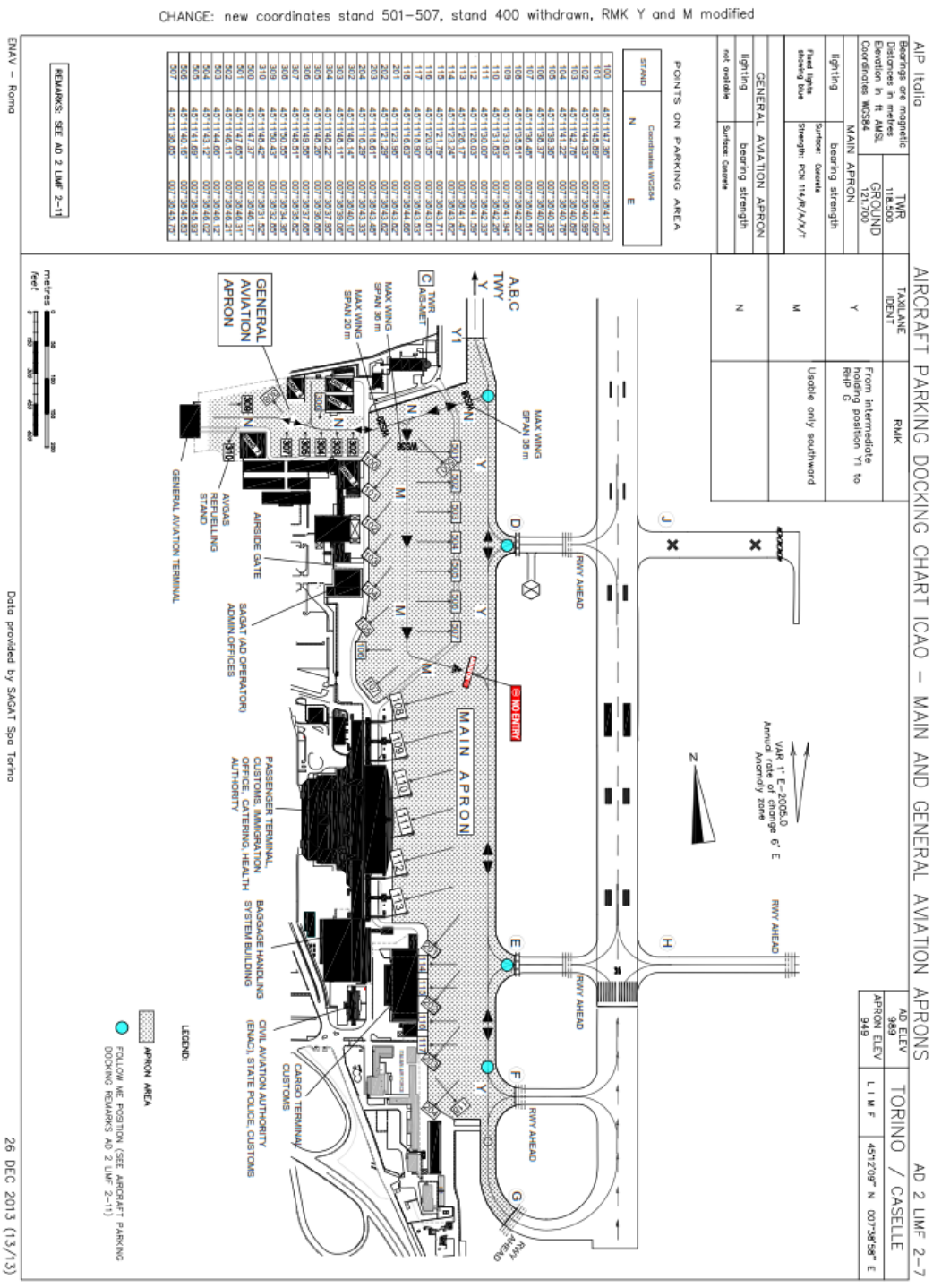


Fig. 4.3 Turin airport layout with indication of the aircraft stands [125].

Three scenarios were analyzed for this airport: the first test case (TC1), presented in Table 4.2, is a simple test case with only one possible runway sequence (refer to

Section 3.1.2 for the definition of runway sequence). The second (TC2) and the third (TC3) test cases have respectively three and seven possible runway sequences (Table 4.3 and 4.4).

Table 4.2 Test Case 1. Flight schedule.

TO/LND	PB/TD time	Stand	RWY entry
LND	14:00:00	113	B
LND	14:11:40	105	A
LND	14:25:40	116	C
TO	14:31:20	113	F
LND	14:36:20	500	B
LND	14:39:40	100	C
TO	14:41:20	105	E
LND	14:49:30	503	A
TO	14:52:40	116	F
LND	14:53:10	506	C

Table 4.3 Test Case 2. Flight schedule.

TO/LND	PB/TD time	Stand	RWY entry
LND	10:00:00	109	C
LND	10:06:50	505	A
LND	10:19:30	112	B
LND	10:30:20	100	C
LND	10:43:10	114	B
TO	10:46:00	109	F
TO	10:52:00	505	E
LND	10:58:00	204	A
TO	11:05:40	114	G
TO	11:09:00	112	F

Table 4.4 Test Case 3. Flight schedule.

TO/LND	PB/TD time	Stand	RWY entry
LND	12:00:00	507	C
LND	12:11:10	203	A
LND	12:25:00	115	B
LND	12:36:30	116	C
LND	12:41:50	117	A
TO	12:43:20	507	F
TO	12:48:10	115	G
TO	12:49:50	203	E
LND	12:56:40	115	C
TO	13:04:00	117	F

4.1.2 Milan Malpensa airport

The Milan Malpensa airport is composed of two separate terminals (T1 and T2). In this work, the runways are considered to be operated in the configuration $35L-35R$ for both departures and arrivals. This airport is used to test the capabilities of the autonomous taxi system of managing more than a fleet of tractors; indeed, two depot areas were defined, one for T1 and one serving T2, as showed in Fig. 4.4. Also for this airport, the airplane stands and the runway entries were extracted from the AIP downloaded from the ENAV website [125].

The flight schedules for the test case scenarios TC4 and TC5 were retrieved from the Milan airport website for what concerns touchdown time, earliest pushback time, and aircraft type assigned to the flights. The corresponding airplane stands and the runway entries were assigned randomly from the airport model. The layout of the airport leads the aircraft landing to or taking off from the runway 35R to cross the runway 35L to reach the terminal T1 or the assigned runway. This feature leads to the definition of a runway exit and a separate connection point in the test case schedule. The first test case (TC4) (Table 4.5) considers that all the airplanes operating in the terminal T1 use runway 35L, whereas the aircraft assigned to the terminal T2 were

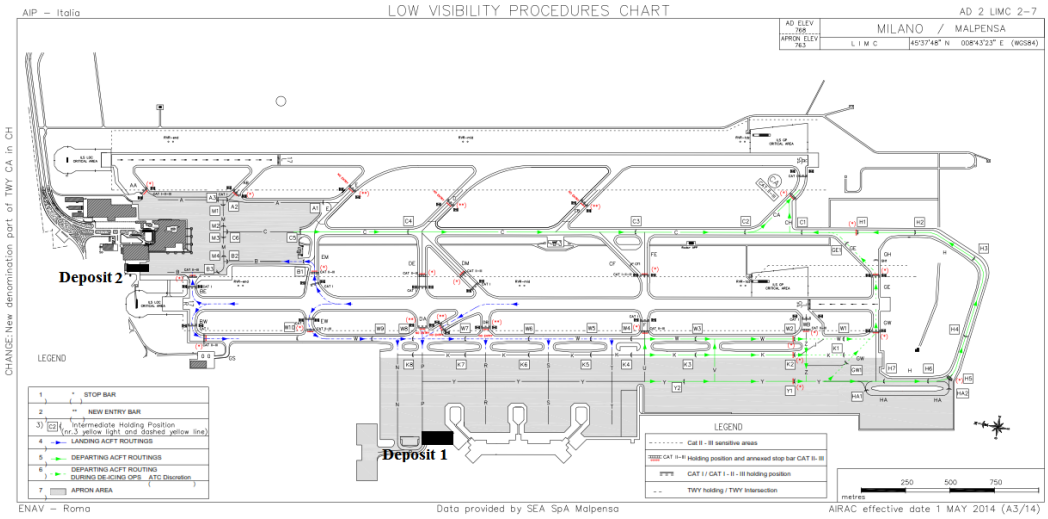


Fig. 4.4 Milan Malpensa airport layout with indication of the depot locations [125].

conceived operating on both runways. Conversely, in the second test case designed for this airport (TC5), presented in Table 4.6, the aircraft assigned to each terminal could operate on every runway.

4.1.3 Amsterdam Schiphol airport

The Amsterdam Schiphol airport is one of the busiest airports in Europe for both passengers and movements. The layout of the airport is really different with respect to the previously considered ones; in fact, the terminals are surrounded by a couple of parallel one-way taxiways: A with clockwise direction and B with counterclockwise direction. The south-west part of the airport is crossed by the two-way taxiway Q. The aeronautical information publication of the Schiphol airport can be accessed at the AIS Netherlands website [126] to have a complete overview of the aerodrome parking locations for each apron and of the ground movement recommendations. Two tractor depot/charging areas were designated for this airport, as described in Fig. 4.5, each one serving a portion of the central apron: the tractors based in the Depot 1 will serve the aircraft that are assigned to the stands inside the red area, whereas the tractors belonging to the Depot 2 will deliver those assigned to the docks within the green area. Runway operations were supposed being in South-North configuration: runways 36L, 06, and 36R are operated by the inbound traffic, whereas runways 36L 36C and 06 are used for the outbound traffic, as reported in [127].

Table 4.5 Test Case 4. Flight schedule.

TO/LND	PB/TD	Stand	A/C code	RWY entry	Conn/Dis	RWY	Terminal
LND	09:00:00	103	319	E	-	35R	2
TO	09:00:00	602	E70	WB	-	35L	1
TO	09:00:00	107	320	CA	-	35R	2
LND	09:05:00	510	321	DB	-	35L	1
LND	09:15:00	601	733	DA	-	35L	1
LND	09:16:10	507	763	EW	-	35L	1
TO	09:20:00	655	319	GW	-	35L	1
TO	09:20:00	609	319	F	-	35L	1
TO	09:25:00	706	733	WB	-	35L	1
TO	09:30:00	105	319	CA	-	35L	2
TO	09:35:00	512	320	GW	-	35L	1
TO	09:40:00	606	736	WB	-	35L	1
TO	09:40:50	604	321	F	-	35L	1
TO	09:41:30	610	320	GW	-	35L	1
LND	09:45:00	662	738	BW	-	35L	1
LND	09:50:00	110	319	E	-	35R	2
LND	09:51:10	651	E90	DA	-	35L	1
TO	09:55:00	552	F70	F	-	35L	1

Table 4.6 Test Case 5. Flight schedule.

TO/LND	PB/TD	Stand	A/C code	RWY entry	Conn/Dis	RWY	Terminal
TO	11:00:00	103	319	CA	-	35R	2
TO	11:00:00	115	319	FE	-	35L	2
TO	11:00:00	707	763	WB	-	35L	1
LND	11:05:00	510	320	E	EW	35R	1
TO	11:05:00	554	738	GW	-	35L	1
LND	11:10:00	601	320	DB	-	35L	1
LND	11:11:00	510	738	EW	-	35L	1
TO	11:10:00	105	320	CA	-	35R	2
TO	11:10:00	113	319	FE	-	35L	2
LND	11:15:00	555	320	DA	-	35L	1
TO	11:20:00	609	320	WB	-	35L	1
TO	11:25:00	501	772	F	-	35L	1
TO	11:30:00	612	320	WB	-	35L	1
TO	11:30:50	552	319	GW	-	35L	1
LND	11:35:00	507	763	E	EW	35R	1
TO	11:35:00	510	320	F	-	35L	1
TO	11:40:00	223	319	GW	-	35L	2
LND	11:45:00	604	320	L	-	35L	1
TO	11:45:00	110	319	CA	-	35R	2
LND	11:50:00	101	319	D	-	35R	2
LND	11:56:30	102	319	E	-	35R	2

Two test cases (TC6 and TC7) were set up for this airport, aimed to test the influence of the depot position on the final solution. Both scenarios are based on the flight schedule presented in Table 4.7. The first test case considers only the presence of Depot 1, thereby concentrating all the traffic flow on it; conversely, the second test case considers both depots, leading to a more distributed traffic. Touchdown time, earliest pushback time, and aircraft type assigned to the flights for the test case scenarios were retrieved from the airport website. The designated airplane stands and the runway entries were randomly defined from the airport model.

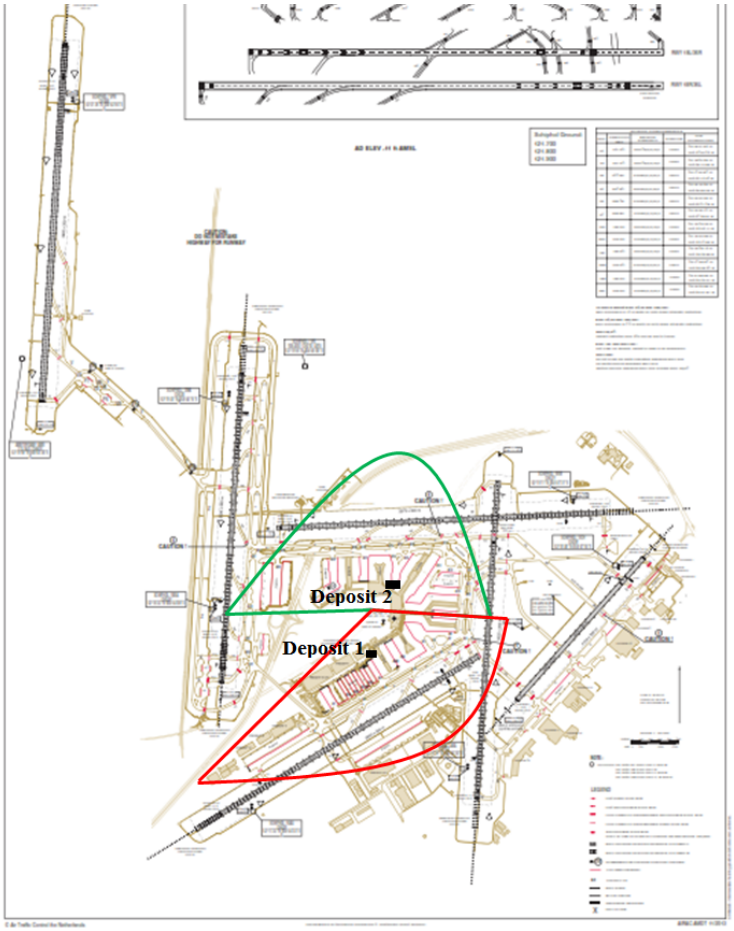


Fig. 4.5 Amsterdam Schiphol airport layout with indication of the the depot locations [126].

Table 4.7 Test Case 6. Flight schedule.

TO/LND	PB/TD	Stand	A/C code	RWY entry	Conn/Dis	RWY	Terminal
LND	07:00:00	G5L	772	E4	-	36R	2
LND	07:00:00	F5L	77W	S5	-	06	2
TO	07:00:00	G7L	319	W5	-	36R	2
TO	07:00:00	D52R	320	S2	-	06	2
TO	07:00:00	E77L	733	S1	-	06	2
TO	07:00:00	H3L	E95	W8	-	36C	2
TO	07:00:00	B93T	320	S1	-	06	1
TO	07:00:00	E9R	E90	V3	Z1	06	1
TO	07:00:00	D93L	E95	S2	-	06	2
TO	07:00:30	F7L	733	W10	-	36C	2
TO	07:00:40	C18L	738	W9	-	36C	1
LND	07:01:20	B36R	332	E2	-	36R	1
LND	07:05:00	E18R	744	S4	-	06	2
TO	07:05:00	E2L	AR8	W10	-	36C	2
TO	07:05:00	D23L	738	S2	-	06	1
TO	07:05:00	B66T	320	W8	-	36C	1
TO	07:05:00	G73R	E75	W9	-	36C	2
TO	07:05:30	C7R	733	S3	-	06	1
TO	07:05:30	J83L	733	S1	-	06	2
LND	07:06:10	H6L	F70	E6	-	36R	2
LND	07:10:00	D49R	F70	E3	-	36R	2
LND	07:10:00	D4L	73H	S3	-	06	1
TO	07:15:00	F3R	E90	V3	W5	36L	2
LND	07:35:40	D18L	333	E2	-	36R	1
LND	07:35:50	J85L	73C	S5	-	06	2
LND	07:40:00	D23L	E90	E6	-	36R	1
LND	07:45:00	A35L	76W	S3	-	06	1
LND	07:50:00	F6L	E95	E4	-	36R	2
LND	07:50:50	C8R	73H	S4	-	06	1
TO	07:51:10	G7L	319	W9	-	36C	2
LND	07:51:10	D52R	E90	E1	-	36R	1
LND	07:51:50	D27L	E75	S4	-	06	1

4.2 Numerical results

Each algorithm developed to solve the trajectory assignment problem was run for each test case and each optimization model $Nr = 100$ times. Afterwards, for the best solution found in the previous step, the tractor dispatch problem was solved running the two proposed algorithms. Models and algorithms are compared in terms of success rate $\%S$, mean and standard deviation of the cost function $\mu_{cost} \pm \sigma_{cost}$, and mean and standard deviation of the computational time $\mu_{time} \pm \sigma_{time}$, for both the trajectory assignment optimizer and the tractor dispatcher. Only the successful runs were considered for the computation of the cost related results.

4.2.1 Trajectory assignment problem

The results for the three test cases in the Turin airport are summarized in Table 4.8 and in Table 4.10 for the continuous and discrete time models respectively and in Fig. 4.6. As it can be noticed, all the algorithms always generate conflict free schedules for the three test cases, when the continuous time model is implemented. The big difference in the cost function between the Tree search algorithm and the two HPSO is given by the different produced scheduling, as described in Table 4.9, which reports the best solution found by each algorithm for TC3 in terms of particle elements divided by flight. The discrete model decreased the reliability of the algorithms leading to lower success rates; Table 4.11 reports the optimal schedule found by each algorithm for TC3, using the discrete time model. The tree search heuristic provides better solutions both from a cost function and a computational time point of view. The solutions obtained with the continuous model resulted in a lower cost function and lower computational time required, with respect to the cases in which the discrete time model was used.

Table 4.12 and Table 4.13 present the results for test cases TC4 and TC5. The trends highlighted for the Turin airport test cases are confirmed both in TC4 and TC5. The RNS-HPSO and the tree search heuristic have generated similar solutions for TC5; however, the TS algorithm ran in less than half time. When the discrete time model is considered, the reliability of the algorithm drastically decreases and the generation of conflict-free schedules can no longer be granted. This behavior can be explained considering that the discrete time model leads the algorithms to overrun the maximum computational time specification, before the algorithms are

Table 4.8 Results for TC1, TC2 and TC3 considering the continuous time model. The results are presented in terms of success rate $\%S$ and mean and standard deviation of the cost function $\mu_{cost} \pm \sigma_{cost}$ and of the computational time required $\mu_{time} \pm \sigma_{time}$.

Algorithm	**Value**	**Test cases**		
		$NF = 10$ $TOs = 3$ $Branches = 1$	$NF = 10$ $TOs = 4$ $Branches = 3$	$NF = 10$ $TOs = 4$ $Branches = 7$
Tree search	$\%S$	100	100	100
	$\mu_{cost} \pm \sigma_{cost}$ $[kWh]$	12.41 ± 0.00	13.30 ± 0.05	13.00 ± 0.00
	$\mu_{time} \pm \sigma_{time}$ $[s]$	1.54 ± 0.03	2.81 ± 0.04	9.69 ± 0.19
HC-HPSO	$\%S$	100	100	100
	$\mu_{cost} \pm \sigma_{cost}$ $[kWh]$	51.05 ± 3.88	55.12 ± 5.22	64.63 ± 11.73
	$\mu_{time} \pm \sigma_{time}$ $[s]$	76.95 ± 31.72	102.08 ± 38.92	79.76 ± 41.94
RNS-HPSO	$\%S$	100	100	100
	$\mu_{cost} \pm \sigma_{cost}$ $[kWh]$	48.36 ± 1.77	51.15 ± 1.38	58.91 ± 9.80
	$\mu_{time} \pm \sigma_{time}$ $[s]$	41.61 ± 13.47	47.60 ± 15.67	50.53 ± 19.26

Table 4.9 Results for TC3 considering the continuous time model. The best found solutions by each algorithm are compared.

Flight	**Tree search**	**HC-HPSO**	**RNS-HPSO**
1	[1, 1, 1, 4, 4, 4, 0, 0, 0]	[1, 1, 1, 4, 4, 4, 0, 0, 0]	[1, 1, 1, 4, 4, 4, 0, 0, 0]
2	[1, 1, 1, 7, 4, 4, 0, 0, 0]	[1, 1, 1, 5, 6.5, 4, 0, 0, 0]	[1, 1, 1, 5, 5, 4, 0, 0, 0]
3	[1, 1, 1, 4, 4, 4, 0, 0, 0]	[1, 1, 1, 6.5, 7, 4, 0, 0, 0]	[1, 1, 1, 12, 7, 4, 0, 0, 0]
4	[1, 1, 1, 4, 4, 4, 0, 0, 0]	[1, 1, 1, 4, 10, 7, 0, 0, 0]	[1, 1, 1, 4, 8, 12, 0, 0, 0]
5	[1, 1, 1, 4, 4, 4, 0, 0, 0]	[1, 1, 1, 4, 8.5, 4, 0, 0, 0]	[1, 1, 1, 4, 9, 4, 0, 0, 0]
6	[1, 1, 1, 4, 4, 4, 0, 0, 0]	[1, 1, 1, 8, 4, 4, 0, 0, 0]	[1, 1, 1, 12, 4, 4, 0, 0, 0]
7	[1, 1, 1, 4, 4, 4, 514, 0, 0]	[1, 1, 1, 4, 7, 4, 0, 0, 0]	[1, 1, 1, 4, 7, 4, 0, 0, 0]
8	[1, 1, 1, 4, 4, 4, 0, 0, 0]	[1, 1, 1, 4, 4, 4, 0, 0, 0]	[1, 1, 1, 4, 4, 4, 0, 0, 0]
9	[1, 1, 1, 4, 4, 4, 0, 0, 0]	[1, 1, 1, 4, 4, 4, 0, 0, 0]	[1, 1, 1, 4, 4, 4, 0, 0, 0]
10	[1, 1, 1, 4, 4, 4, 0, 0, 0]	[1, 1, 1, 4, 4, 4, 0, 0, 0]	[1, 1, 1, 4, 4, 4, 0, 0, 0]

capable of reaching a feasible solution. Indeed, the only algorithm capable of finding conflict-free schedules for test case 4 (HC-HPSO) presented a mean computational time higher that 15 *min*.

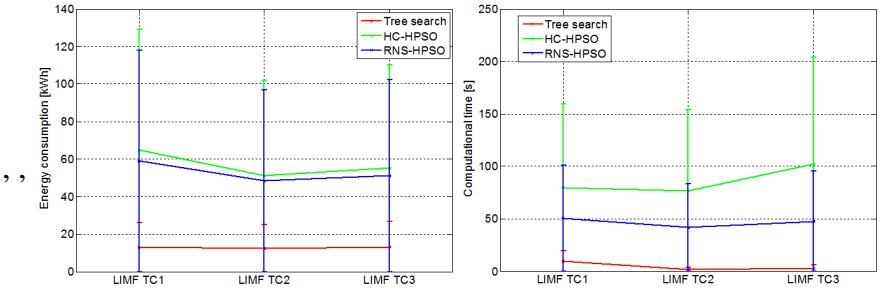
Table 4.10 Results for TC1, TC2 and TC3 considering the discrete time model. The results are presented in terms of success rate $\%S$, mean and standard deviation of the cost function $\mu_{cost} \pm \sigma_{cost}$ and of the computational time required $\mu_{time} \pm \sigma_{time}$.

Algorithm	**Value**	**Test cases**		
		$NF = 10$ $TOs = 3$ $Branches = 1$	$NF = 10$ $TOs = 4$ $Branches = 3$	$NF = 10$ $TOs = 4$ $Branches = 7$
Tree search	$\%S$	100	74	97
	$\mu_{cost} \pm \sigma_{cost}$ $[kWh]$	46.47 ± 0.08	51.32 ± 1.03	51.01 ± 0.58
	$\mu_{time} \pm \sigma_{time}$ $[s]$	7.01 ± 0.48	49.37 ± 24.35	106.65 ± 109.82
HC-HPSO	$\%S$	100	100	100
	$\mu_{cost} \pm \sigma_{cost}$ $[kWh]$	49.69 ± 3.28	55.88 ± 3.75	65.01 ± 8.33
	$\mu_{time} \pm \sigma_{time}$ $[s]$	531.51 ± 232.41	427.78 ± 197.41	395.14 ± 178.32
RNS-HPSO	$\%S$	96	92	46
	$\mu_{cost} \pm \sigma_{cost}$ $[kWh]$	76.41 ± 14.15	87.34 ± 15.37	101.75 ± 12.17
	$\mu_{time} \pm \sigma_{time}$ $[s]$	148.38 ± 63.03	143.39 ± 73.55	413.55 ± 198.99

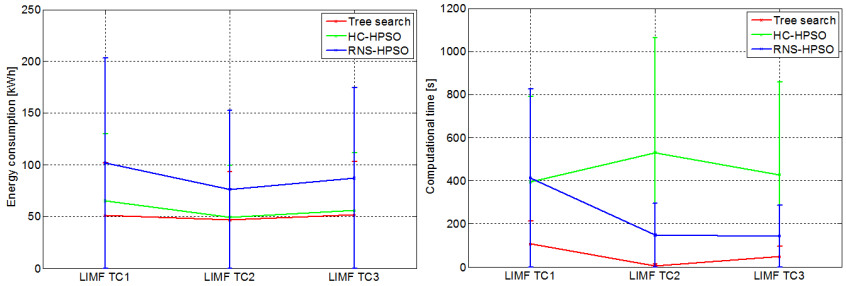
Table 4.11 Results for TC3 considering the discrete time model. The best found solutions by each algorithm are compared.

Flight	**Tree search**	**HC-HPSO**	**RNS-HPSO**
1	[1, 1, 1, 4, 4, 4, 0, 0, 0]	[1, 1, 1, 4, 4, 15, 0, 0, 24]	[1, 1, 1, 4, 4, 4, 0, 3, 0]
2	[1, 1, 1, 4, 4, 4, 0, 0, 0]	[1, 1, 1, 4, 9, 4, 9, 21, 38]	[1, 1, 1, 6, 4, 12, 0, 0, 5]
3	[1, 1, 1, 4, 4, 4, 0, 600, 0]	[1, 1, 1, 6.5, 4, 9.5, 1, 0, 15]	[1, 1, 1, 4, 5, 4, 50, 0, 20]
4	[1, 1, 1, 4, 4, 4, 0, 230, 0]	[1, 1, 1, 10, 4, 13, 0, 2, 80]	[1, 1, 1, 4, 6, 8, 90, 12, 30]
5	[1, 1, 1, 4, 4, 4, 0, 0, 0]	[1, 1, 1, 8.5, 4, 5.5, 0, 0, 19]	[1, 1, 1, 13, 4, 8, 0, 0, 90]
6	[1, 1, 1, 4, 4, 4, 0, 0, 210]	[1, 1, 1, 12, 4, 4, 3, 48, 47]	[1, 1, 1, 4, 6, 4, 0, 0, 0]
7	[1, 1, 1, 4, 6, 4, 0, 0, 90]	[1, 1, 1, 4, 10, 4, 0, 0, 84]	[1, 1, 1, 4, 4, 4, 43, 90, 42]
8	[1, 1, 1, 4, 4, 4, 0, 530, 0]	[1, 1, 1, 4, 4, 4, 1, 43, 52]	[1, 1, 1, 7, 4, 8, 1, 45, 0]
9	[1, 1, 1, 4, 4, 4, 350, 0, 0]	[1, 1, 1, 4, 4, 4.5, 24, 0, 0]	[1, 1, 1, 4, 4, 4, 43, 0, 62]
10	[1, 1, 1, 4, 4, 4, 0, 0, 170]	[1, 1, 1, 4, 4, 4, 49, 0, 0]	[1, 1, 1, 5, 4, 4, 8, 19, 0]

The results concerning TC6 and TC7 are reported in Table 4.14. For these test cases, only the runs using the continuous model are proposed, as none of the algorithms was able to converge to feasible solutions when using the discrete time model. Also considering the test cases related to the Amsterdam Schiphol airport,



(a) Continuous time model



(b) Discrete time model

Fig. 4.6 LIMF test cases TC1, TC2 and TC3. Mean cost function and mean computational time for the TS, HC-HPSO and RNS-HPSO algorithms

the tree search heuristic resulted the most effective and efficient algorithm, leading to less expensive solutions in a dramatically lower time, with respect to the two hybridizations of the particle swarm optimization. It is possible to notice that the solution found for TC7 are really close to the ones found for TC6; therefore, the position of the depot has not deeply influenced the overall consumption. For these test cases, both the HC-HPSO and the RNS-HPSO violate the maximum computational time constraint.

Table 4.12 Results for TC4 and TC5 considering the continuous time model. The results are presented in terms of success rate $\%S$ and mean and standard deviation of the cost function $\mu_{cost} \pm \sigma_{cost}$ and of the computational time required $\mu_{time} \pm \sigma_{time}$.

Algorithm	**Value**	**Test cases** $NF = 18$, $TOs = 11$, $Branches = 146$	**Test cases** $NF = 20$, $TOs = 13$, $Branches = 56$
Tree search	$\%S$	100	100
	$\mu_{cost} \pm \sigma_{cost}$	112.26 ± 0.24	172.78 ± 0.00
	$\mu_{time} \pm \sigma_{time}$	264.76 ± 8.10	196.82 ± 14.37
HC-HPSO	$\%S$	100	100
	$\mu_{cost} \pm \sigma_{cost}$	179.24 ± 23.89	252.60 ± 25.29
	$\mu_{time} \pm \sigma_{time}$	424.66 ± 207.56	755.80 ± 223.77
RNS-HPSO	$\%S$	100	100
	$\mu_{cost} \pm \sigma_{cost}$	189.48 ± 21.11	172.51 ± 5.15
	$\mu_{time} \pm \sigma_{time}$	396.64 ± 169.36	503.62 ± 227.85

Table 4.13 Results for TC4 and TC5 considering the discrete time model. The results are presented in terms of success rate $\%S$ and mean and standard deviation of the cost function $\mu_{cost} \pm \sigma_{cost}$ and of the computational time required $\mu_{time} \pm \sigma_{time}$. The text highlighted in red indicates that the computational time is higher than the maximum time allowed.

Algorithm	**Value**	**Test cases** $NF = 18$, $TOs = 11$, $Branches = 146$	**Test cases** $NF = 20$, $TOs = 13$, $Branches = 56$
Tree search	$\%S$	0	0
	$\mu_{cost} \pm \sigma_{cost}$	–	–
	$\mu_{time} \pm \sigma_{time}$	–	–
HC-HPSO	$\%S$	64	0
	$\mu_{cost} \pm \sigma_{cost}$	246.33 ± 14.20	–
	$\mu_{time} \pm \sigma_{time}$	1210.49 ± 57.50	–
RNS-HPSO	$\%S$	0	0
	$\mu_{cost} \pm \sigma_{cost}$	–	–
	$\mu_{time} \pm \sigma_{time}$	–	–

Table 4.14 Results for TC6 and TC7 considering the continuous time model. The results are presented in terms of success rate $\%S$ and mean and standard deviation of the cost function $\mu_{cost} \pm \sigma_{cost}$ and of the computational time required $\mu_{time} \pm \sigma_{time}$. The text highlighted in red indicates that the computational time is higher than the maximum time allowed.

Algorithm	**Value**	**Test cases**	
		$NF = 32$ $TOs = 17$ $Branches = 30$	$NF = 32$ $TOs = 17$ $Branches = 30$
Tree search	$\%S$	100	100
	$\mu_{cost} \pm \sigma_{cost}$	311.90 ± 0.72	294.89 ± 2.74
	$\mu_{time} \pm \sigma_{time}$	277.74 ± 4.81	360.62 ± 288.89
HC-HPSO	$\%S$	7	24
	$\mu_{cost} \pm \sigma_{cost}$	404.53 ± 16.90	373.52 ± 43.50
	$\mu_{time} \pm \sigma_{time}$	925.34 ± 9.30	922.08 ± 12.13
RNS-HPSO	$\%S$	1	27
	$\mu_{cost} \pm \sigma_{cost}$	322.75 ± 12.78	306.12 ± 10.32
	$\mu_{time} \pm \sigma_{time}$	911.79 ± 14.56	919.17 ± 11.23

4.2.2 Tractor dispatch

For each of the test cases presented above, given the best solution found by the algorithms, the minimum number of tugs needed to perform the schedule was computed by means of the tractor dispatch algorithm. Starting from a fleet size equal to the number of flights in the flight schedule, the value NT was progressively decreased until no feasible assignments were generated. The results are reported in Table 4.15. It descends that in the Turin airport at least 5 tractors are required; as far as the Milan airport is considered, a minimum of 8 tugs have to be allocated to the Terminal 1, whereas at least 4 to the Terminal 2. The minimum number of tractors required to perform the proposed test cases in the Amsterdam airport decreases if two depots are considered. This can be explained by the fact that in case two depots are available, the travel time required to reach the airplanes or to go back to the depot is lower. This feature allows to increase the number of missions that a tug can perform during the same time period.

Table 4.15 Minimum number of tractors to be allocated to each depot, in order to perform the schedules generated by the trajectory assignment algorithm

Airport	TC	NT	
		Depot 1	Depot 2
LIMF	TC1	5	–
	TC2	3	–
	TC3	5	–
LIMC	TC4	8	4
	TC5	7	4
EHAM	TC6	26	–
	TC7	10	14

4.3 Comparison with classical taxi

This section presents a comparison between the total estimated operative costs of performing the optimized tug schedule (engine-off) and an estimation of the total cost for the classical taxi (engine-on), aimed at the evaluation of the potential savings deriving from the adoption of the autonomous taxi system. The approach proposed in [29] and [128] is used. When the taxi phase is performed with the main engines off, the airplane systems need to be powered by the auxiliary power unit (APU), thereby implying a cost that has to be added to the energy consumed by the tugs for a more realistic cost estimation. The considered average APU fuel flow is 0.03 kg/s [129], with a fuel cost of 0.5254 $USD/kg = 0.4938\ EUR/kg$ ($1USD = 0.94EUR$) [130]. The average energy cost in Europe for industrial applications during the first semester of the 2016 was 0.117 EUR/kWh [3].

For each departure/arrival, the fuel consumed using the classical taxi approach is computed using the following equation [128]:

$$\frac{f_{cons}}{\sqrt{T_{amb}}} = a_f + b_f \cdot t + c_f \cdot n_{acc} \tag{4.3}$$

where f_{cons} is the total fuel consumed, $T_{amb} = 15\ ^{\circ}C = 288.15\ K$ is the external temperature, t is the traveling time and n_a is the number of acceleration events. The values of the coefficients a_f, b_f, c_f for each airplane were retrieved from [128]. Given the limited amount of airplane coefficient available, the same data were used also for airplanes not analyzed in [128] but having similar characteristics to the ones for which the data are available. In this work, the number of acceleration events was set to $n_{acc} = 1$ (only the first acceleration considered), thereby assuming that the aircraft would be able to perform the taxi trajectory continuously at constant speed. Furthermore, the aircraft are considered taking off as soon as they reach the runway, without any queue delay. This assumptions leads to underestimated taxi consumptions for the engine-on approach, thereby to a more conservative saving evaluation.

For each test case, the classical taxi cost is evaluated for two different taxiing speed levels: $V = 10\ m/s$ and $V = 16\ m/s$. In this way, it is possible to define a range of savings and to detect if any situation occurs in which the classical taxi is more convenient than the autonomous one. The results of the analysis are reported in Table 4.16. As far as the towing phase is considered, only the portion of trajectory from the stand to the disconnection point or from the connection point to the stand is accounted, thereby neglecting the portions of trajectory traveled by airplanes with own propulsion; same approach was used for the engines-on taxi. It can be noticed that the adoption of an autonomous taxi system always results more convenient, with respect to the engine-on taxi. The estimated savings are in the range $40\% - 80\%$, with a significant increase according to the airport size; this tendency leads to infer that the bigger the airport, the higher the potential savings.

Table 4.16 Costs related to the autonomous or classical taxi solution, expressed in EUR. The savings are indicated as percentage of the engine-on taxi cost.

TC	Autonomous taxi	Classical taxi			
		V=10 m/s	Savings %	V=16 m/s	Savings %
TC1	54.83	152.35	64.01	96.57	43.22
TC2	46.59	162.14	71.27	102.62	54.60
TC3	58.18	161.34	63.94	102.12	43.02
TC4	114.87	396.09	71.00	260.53	55.91
TC5	144.57	600.51	75.92	386.08	62.55
TC6	249.95	1321.40	81.08	849.45	70.57
TC7	298.35	1321.40	77.42	849.45	64.88

Chapter 5

Conclusions

This dissertation presents the activities performed during the research project. Through the chapters, it is possible to find a combination of computational efficiency topics and physical modeling. The first part of the work focused on the development of models and tools to provide a physical representation of the autonomous taxi problem; furthermore, the mathematical formulation of the two main problems, the trajectory assignment and departure sequencing problem and the tug dispatch problem, have been proposed, together with the NP-hardness proof.

The formulation of combinatorial optimization schemes dominated the second part of the work. Some aspects of the hybrid particle swarm optimization algorithms can be improved. In fact, only some portions of the code are parallelized, while a more extensive parallelization could lead to lower computational time, thereby improving the algorithm efficiency and also its reliability. As far as the tree search algorithm is concerned, the number of feasible runway sequences can become very large, slowing down the algorithm. A function capable of determining if a runway sequence is promising, compared to the optimal solution found so far, should be implemented to avoid unnecessary computations.

Simulation results showed that the discrete time based model negatively alters the optimization algorithms performance, both in terms of effectiveness and efficiency. Therefore, the continuous time model will be preferred in future developments. Moreover, the tree search heuristic has proven to be more effective than the two hybrid versions of the particle swarm optimization, generating, for all the test cases, conflict-free schedules with lower cost function value in a lower computational time.

Finally, the cost comparison between the classical engine-on taxi and the autonomous version showed that the proposed solution is always more convenient; furthermore, the difference between the two approaches increases as the airport size increases.

The developed models consider a deterministic framework, in which the pushback time and touch down time, defined in the flight schedule, are constant. However, delays both during take-off and landing might occur; therefore, future work should focus on the implementation of stochastic models and on adapting the proposed optimization schemes to solve stochastic taxi problems.

Appendix A

Airport discretization GUI

The graphical user interface showed below uses the pre-compiled function *plot_google_map* [131] to show the map of the chosen airport. If a discretization grid is present in the airport file, it can be displayed on the map by clicking the *Plot Grid* button; nodes will be plotted in cyan, while connections are represented by arrow showing the possible directions. New nodes can be created using the central panel of the GUI, called *Nodes Definition*; the *New Nodes* button calls the *getpts* function, which allows to catch some points coordinates on the map using the mouse. Some tips on the *getpts* usage are displayed on the right top of the GUI when the function is activated. The same function is used to delete nodes: the user clicks the mouse left button over/near the desired node, and an algorithm recognizes the right node and delete it. Any connection starting from or arriving to that node will be deleted. The user has also the possibility to delete all nodes with the last button of this panel, called *Delete all Nodes*.

The right panel, called *Connectivity Definition*, can be used to add or delete connections between nodes. The first button is a toggle button which enables to define one-way connections, or to delete them only in one direction. As default the two-way connections are considered. When the *New Connections* button is pressed, the *getline* function is called, which allows to draw a line between to points and catches the starting and the final point locations. A series of connections can also be defined by clicking on a series of consecutive nodes. The *Delete Connections* button works in the same manner of the previous one, whereas the button called *Delete all Connections* deletes all the connections at the same time.

The bottom right buttons can be used to assign to each node a feature (i.e. stand ID, runway entry ID) and to create the runway objects. Finally, the *Save* and the *Write C* buttons are used to create the output in MATLAB or .txt file format respectively.

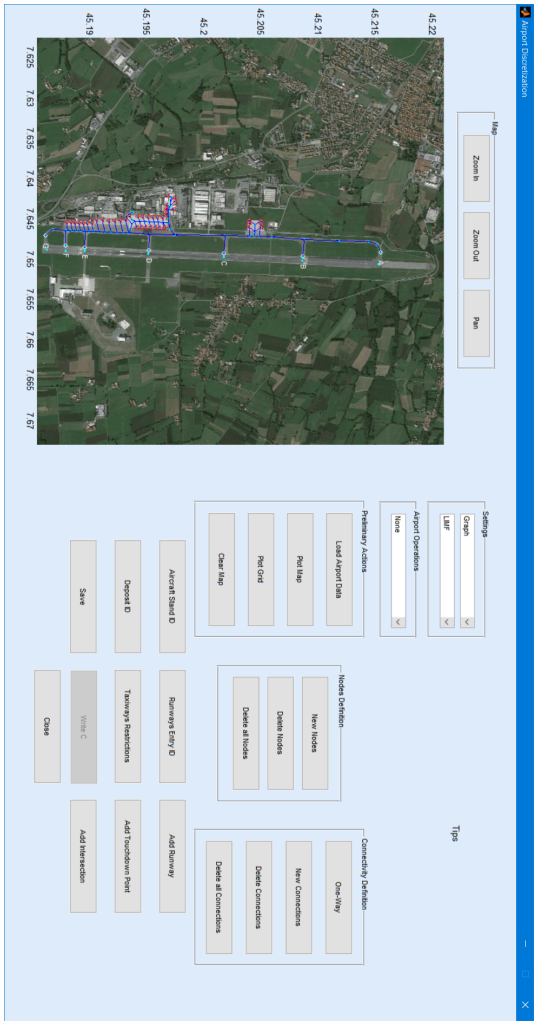


Fig. A.1 Airport discretization graphical user interface (GUI). Credits: Pictures 2015 DigitalGlobe, Map data ©2015 Google.

References

[1] Airport Council International. Aci website, 2016. Accessed: 27/10/2016.

[2] Bureau Of Transportation Statistics. Bts website, 2016. Accessed: 23/11/2016.

[3] European commission. Eurostat website, 2016. Accessed: 23/11/2016.

[4] FAA. Aviation system performance metrics, 2017. Accessed: 17/01/2017.

[5] I. Simaiakis and H. Balakrishnan. Impact of congestion on taxi times, fuel burn, and emissions at major airports. *Transportation Research Record: Journal of the Transportation Research Board*, (2184):22–30, 2010.

[6] National Transportation Safety Board. National transportation safety board website, 2016. Accessed: 02/11/2016.

[7] Cynthia Barnhart, Peter Belobaba, and Amedeo R Odoni. Applications of operations research in the air transport industry. *Transportation science*, 37(4):368–391, 2003.

[8] Ioannis Anagnostakis, John-Paul Clarke, Dietmar Böhme, and Uwe Völckers. Runway operations planning and control: Sequencing and scheduling. *Journal of Aircraft*, 38(6):988–996, 2001.

[9] Nicolas Pujet, Bertrand Delcaire, and Eric Feron. Input-output modeling and control of the departure process of busy airports. *Air Traffic Control Quarterly*, 8(1):1–32, 2000.

[10] Kari Anderson, Francis Carr, Eric Feron, and William Hall. Analysis and modeling of ground operations at hub airports. Technical report, 2000.

[11] Jean-Baptiste Gotteland, Nicolas Durand, Jean-Marc Alliot, and Erwan Page. Aircraft ground traffic optimization. In *ATM 2001, 4th USA/Europe Air Traffic Management Research and Development Seminar*, pages pp–xxxx, 2001.

[12] J-B Gotteland and Nicolas Durand. Genetic algorithms applied to airport ground traffic optimization. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 1, pages 544–551. IEEE, 2003.

[13] Nicolas Durand, Cyril Allignol, and Nicolas Barnier. A ground holding model for aircraft deconfliction. In *Digital Avionics Systems Conference (DASC), 2010 IEEE/AIAA 29th*, pages 2–D. IEEE, 2010.

[14] J. Guépet, O. Briant, J.P. Gayon, and R. Acuna-Agost. The aircraft ground routing problem: Analysis of industry punctuality indicators in a sustainable perspective. *European Journal of Operational Research*, 248(3):827 – 839, 2016.

[15] C. Liu and K. Guo. Airport taxi scheduling optimization based on genetic algorithm. In *Computational Intelligence and Security (CIS), 2010 International Conference on*, pages 205–208, Dec 2010.

[16] J. Yu, L. Zhihua, and Z. Honghai. A collaborative optimization model for ground taxi based on aircraft priority. *Mathematical Problems in Engineering*, pages 1 – 9, 2013.

[17] Z. Jingnan and L. Qing. Airport taxi scheduling strategy based on particle swarm optimization algorithm. In *International Conference on Mechatronics, Control and Electronic Engineering (MCE 2014)*, 2014.

[18] S. Ravizza, J. A. D. Atkin, and E. K. Burke. A more realistic approach for airport ground movement optimisation with stand holding. *Journal of Scheduling*, 17(5):507–520, 2014.

[19] C. Stergianos, J. Atkin, P. Schittekat, T. Nordlander, C. Gerada, and H. Morvan. Pushback delays on the routing and scheduling problem of aircraft. *Lecture Notes in Management Science*, 7:34 – 40, 2015.

[20] H. Zhou and X. Jiang. Research on taxiway path optimization based on conflict detection. *PLoS ONE*, 10(7):1–17, 07 2015.

[21] M. Weiszer, J. Chen, S. Ravizza, J. Atkin, and P. Stewart. A heuristic approach to greener airport ground movement. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 3280–3286, July 2014.

[22] M. Weiszer, J. C., and P. Stewart. A real-time active routing approach via a database for airport surface movement. *Transportation Research Part C: Emerging Technologies*, 58, Part A:127 – 145, 2015.

[23] J. Chen, M. Weiszer, P. Stewart, and M. Shabani. Toward a more realistic, cost-effective, and greener ground movement through active routing part i: Optimal speed profile generation. *IEEE Transactions on Intelligent Transportation Systems*, 17(5):1196–1209, May 2016.

[24] T. Zhang, M. Ding, H. Zuo, L. Zeng, and Z. Sun. A two-stage airport ground movement speed profile design methodology using particle swarm optimization. *International Journal of Mathematical, Computational, Physical, Electrical and Computer Engineering*, 9(9):15 – 21, 2015.

[25] T. Zhang, M. Ding, H. Zuo, L. Zeng, and Z. Sun. A two-stage speed profile design methodology for smooth and fuel efficient aircraft ground movement. In *2016 12th IEEE International Conference on Control and Automation (ICCA)*, pages 479–484, June 2016.

[26] D. Kjenstad, C. Mannino, T. E. Nordlander, P. Schittekat, and M. Smedsrud. Optimizing aman-sman-dman at hamburg and arlanda airport. *Proceedings of the SID, Stockholm*, 2013.

[27] Honeywell International Inc. Honeywell international inc. website, 2016. Accessed: 31/10/2016.

[28] WheelTug plc. Wheeltug plc website, 2016. Accessed: 31/10/2016.

[29] J. Hospodka. Cost-benefit analysis of electric taxi systems for aircraft. *Journal of Air Transport Management*, 39:81 – 88, 2014.

[30] Eric V Denardo. *Dynamic programming: models and applications*. Courier Corporation, 2012.

[31] Lufthansa. Innovative taxibot now used in real flight operations. 2015.

[32] M. Battipede, A. Della Corte, M. Vazzola, and D. Tancredi. Innovative airplane ground handling system for green operations. In 27^{th} *International Congress Of The Aeronautical Sciences ICAS*, 2010.

[33] J. Y. Kim, K. Aktay, K. Aktay, and T. D. Ropp. Ants-automated nextgen taxi system. 2010.

[34] Z. K. Chua. Self-managing conflict resolution for autonomous taxiing tugs: An initial survey. *Image*, 17:2, 2015.

[35] R. Morris, M. Lee Chang, R. Archer, E. Cross, S. Thompson, J. Franke, R. Garrett, W. Malik, K. McGuire, and G. Hemann. Self-driving aircraft towing vehicles: A preliminary report, 2015.

[36] M. Ghallab, D. Nau, and P. Traverso. *Automated planning: theory & practice*. Elsevier, 2004.

[37] H. Geffner and B. Bonet. A concise introduction to models and methods for automated planning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 8(1):1–141, 2013.

[38] E. L Lawler, J. K. Lenstra, A. HG R. Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. *Handbooks in operations research and management science*, 4:445–522, 1993.

[39] K. R. Baker and D. Trietsch. *Principles of sequencing and scheduling*. John Wiley & Sons, 2013.

[40] A. D'Ariano, D. Pacciarelli, and M. Pranzo. A branch and bound algorithm for scheduling trains in a railway network. *European Journal of Operational Research*, 183(2):643 – 657, 2007.

[41] H. J. Kim, J. H. Lee, and T. E. Lee. Noncyclic scheduling of cluster tools with a branch and bound algorithm. *IEEE Transactions on Automation Science and Engineering*, 12(2):690–700, 2015.

[42] C. Chandra, Z. Liu, J. He, and T. Ruohonen. A binary branch and bound algorithm to minimize maximum scheduling cost. *Omega*, 42(1):9 – 15, 2014.

[43] S. Wang, M. Liu, and C. Chu. A branch-and-bound algorithm for two-stage no-wait hybrid flow-shop scheduling. *International Journal of Production Research*, 53(4):1143–1167, 2015.

[44] C.-F. Liaw. An improved branch-and-bound algorithm for the preemptive open shop total completion time scheduling problem. *Journal of Industrial and Production Engineering*, 30(5):327–335, 2013.

[45] J. García, J. E. Florez, Á. Torralba, D. Borrajo, C. L. López, Á. García-Olaya, and J. Sáenz. Combining linear programming and automated planning to solve intermodal transportation problems. *European Journal of Operational Research*, 227(1):216 – 226, 2013.

[46] Y.-F. Hung and R. C. Leachman. A production planning methodology for semiconductor manufacturing based on iterative simulation and linear programming calculations. *IEEE Transactions on Semiconductor Manufacturing*, 9(2):257–269, May 1996.

[47] H. Li and N. K. Womer. Solving stochastic resource-constrained project scheduling problems by closed-loop approximate dynamic programming. *European Journal of Operational Research*, 246(1):20 – 33, 2015.

[48] S. Im, S. Li, B. Moseley, and E. Torng. A dynamic programming framework for non-preemptive scheduling problems on multiple machines. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1070–1086. SIAM, 2015.

[49] J. A.S. Gromicho, J. J. van Hoorn, F. Saldanha da Gama, and G. T. Timmer. Solving the job-shop scheduling problem optimally by dynamic programming. *Computers and Operations Research*, 39(12):2968 – 2977, 2012.

[50] M. F. Tasgetiren, Q.-K. Pan, P.N. Suganthan, and O. Buyukdagli. A variable iterated greedy algorithm with differential evolution for the no-idle permutation flowshop scheduling problem. *Computers & Operations Research*, 40(7):1729 – 1743, 2013.

[51] F. J. Rodriguez, M. Lozano, C. Blum, and C. García-Martínez. An iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem. *Computers & Operations Research*, 40(7):1829 – 1841, 2013.

[52] J. E. Beasley, J. Sonander, and P. Havelock. Scheduling aircraft landings at London Heathrow using a population heuristic. *Journal of the Operational Research Society*, 52(5):483–493, 2001.

[53] S. Khebbache-Hadji, C. Prins, A. Yalaoui, and M. Reghioui. Heuristics and memetic algorithm for the two-dimensional loading capacitated vehicle routing problem with time windows. *Central European Journal of Operations Research*, 21(2):307–336, 2011.

[54] S. Mitrović-Minić and G. Laporte. Waiting strategies for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(7):635–655, 2004.

[55] R.A. Gallego, R. Romero, and A.J. Monticelli. Tabu search algorithm for network synthesis. *IEEE Transactions on Power Systems*, 15(2):490–495, 2000.

[56] M. Gendreau, F. Guertin, J.-Y. Potvin, and E. Taillard. Parallel Tabu Search for Real-Time Vehicle Routing and Dispatching. *Transportation Science*, 33(4):381–390, 1999.

[57] J. Brandão and A. Mercer. A tabu search algorithm for the multi-trip vehicle routing and scheduling problem. *European Journal of Operational Research*, 100(1):180–191, 1997.

[58] D. B. Skalak. Prototype and feature selection by sampling and random mutation hill climbing algorithms. *Baseline*, pages 1–15, 1994.

[59] A. Lim, B. Rodrigues, and X. Zhang. A simulated annealing and hill-climbing algorithm for the traveling tournament problem. *European Journal of Operational Research*, 174(3):1459–1478, 2006.

[60] C. Blum. Beam-aco—hybridizing ant colony optimization with beam search: an application to open shop scheduling. *Computers and Operations Research*, 32(6):1565 – 1591, 2005.

[61] J. E. Bell and P. R. McMullen. Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics*, 18(1):41 – 48, 2004.

[62] R. Nakano and T. Yamada. Conventional Genetic Algorithm for Job Shop Problems, 1991.

[63] S. Liu, W. Huang, and H. Ma. An effective genetic algorithm for the fleet size and mix vehicle routing problems. *Transportation Research Part E: Logistics and Transportation Review*, 45(3):434–445, 2009.

[64] B. M. Baker and M.a. Ayechew. A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5):787–800, 2003.

[65] M. Bielli, M. Caramia, and P. Carotenuto. Genetic algorithms in bus network optimization. *Transportation Research Part C: Emerging Technologies*, 10(1):19–34, 2002.

[66] A. Salman, I. Ahmad, and S. Al-Madani. Particle swarm optimization for task assignment problem. *Microprocessors and Microsystems*, 26(8):363–371, 2002.

[67] S. Pandey, L. Wu, S. M. Guru, and R. Buyya. A particle swarm optimization-based heuristic for scheduling workflow applications in cloud computing environments. *Proceedings - International Conference on Advanced Information Networking and Applications, AINA*, (July):400–407, 2010.

[68] C. A. Persiani, F. Donello, and T. Bombardi. Aircraft sequencing problem via two-stage particle swarm optimization. In *ASDA Seminar, Toulouse*, July 2013.

[69] J. Ding and Y. Zhang. A discrete particle swarm optimization algorithm for gate and runway combinatorial optimization problem. *Research Journal of Applied Sciences, Engineering and Technology*, 5(10), 2013.

[70] B. S. Girish. An efficient hybrid particle swarm optimization algorithm in a rolling horizon framework for the aircraft landing problem. *Applied Soft Computing*, 44:200 – 221, 2016.

[71] Z. Michalewicz and D. B. Fogel. *How to Solve It: Modern Heuristics*. Springer, enlarged 2nd edition, December 2004.

[72] C. Blum, J. Puchinger, G. R. Raidl, and A. Roli. Hybrid metaheuristics in combinatorial optimization: A survey. *Applied Soft Computing*, 11(6):4135 – 4151, 2011.

[73] R. C. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the sixth international symposium on micro machine and human science*, volume 1, pages 39–43. New York, NY, 1995.

[74] M. Clerc. Discrete particle swarm optimization, illustrated by the traveling salesman problem. In *New Optimization Techniques in Engineering*, volume 141 of *Studies in Fuzziness and Soft Computing*, pages 219–239. Springer Berlin Heidelberg, 2004.

[75] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73, Feb 2002.

[76] I. C. Trelea. The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information Processing Letters*, 85(6):317 – 325, 2003.

[77] R. Hassan, B. Cohanim, O. De Weck, and G. Venter. A comparison of particle swarm optimization and the genetic algorithm. In *Proceedings of the 1st AIAA multidisciplinary design optimization specialist conference*, pages 18–21, 2005.

[78] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[79] G. Sirigu, M. Battipede, J.-P. Clarke, and P. Gili. A fleet management algorithm for automatic taxi operations. In *ICRAT 2016 7th International Conference on Research in Air Transportation*, pages 1–5, 2016.

[80] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[81] M.H. Xu, Y.Q. Liu, Q.L. Huang, Y.X. Zhang, and G.F. Luan. An improved dijkstra's shortest path algorithm for sparse network. *Applied Mathematics and Computation*, 185(1):247 – 254, 2007.

[82] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on*, 4(2):100–107, July 1968.

[83] W. Zeng and R. L. Church. Finding shortest paths on real road networks: the case for a*. *International Journal of Geographical Information Science*, 23(4):531–543, 2009.

[84] C.Y. Lee. An algorithm for path connections and its applications. *Electronic Computers, IRE Transactions on*, EC-10(3):346–365, Sept 1961.

[85] S. S. Skiena. *The Algorithm Design Manual*. Springer Publishing Company, Incorporated, 2nd edition, 2008.

[86] Giuseppe Sirigu, Mario Cassaro, Manuela Battipede, and Piero Gili. A route selection problem applied to auto-piloted aircraft tugs. *WSEAS TRANSACTIONS ON ELECTRONICS*, 8:27–40, 2017.

[87] J. Y. Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

[88] D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1998.

[89] H. Aljazzar and S. Leue. K∗: A heuristic search algorithm for finding the k shortest paths. *Artificial Intelligence*, 175(18):2129 – 2154, 2011.

[90] J. Van Roy, N. Leemput, S. De Breucker, F. Geth, P. Tant, and J. Driesen. An availability analysis and energy consumption model for a flemish fleet of electric vehicles. *status: published*, 2011.

[91] J. Larminie and J. Lowry. *Electric Vehicle Technology Explained*. Wiley, 2003.

[92] R. Maia, M. Silva, R. Araujo, and U. Nunes. Electric vehicle simulator for energy consumption studies in electric mobility systems. In *Integrated and Sustainable Transportation System (FISTS), 2011 IEEE Forum on*, pages 227–232, June 2011.

[93] ICAO. *Aerodromes: Annex 14 to the Convention on International Civil Aviation. Aerodrome design and operations*. Number Vol. 1 in ICAO Annex. 2013.

[94] Henrik Olsson, Karl J Åström, C Canudas De Wit, Magnus Gäfvert, and Pablo Lischinsky. Friction models and friction compensation. *European journal of control*, 4(3):176–195, 1998.

[95] P. C. Roling and H. G. Visser. Optimal airport surface traffic planning using mixed-integer linear programming. *Int. J. Aero. Eng.*, 2008(1):1:1–1:11, January 2008.

[96] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: A hierarchical structure for rapid interference detection. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pages 171–180, New York, NY, USA, 1996. ACM.

[97] Z. Jia, A. Gallagher, A. Saxena, and T. Chen. 3d-based reasoning with blocks, support, and stability. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[98] M. P. Nepal, S. Staub-French, R. Pottinger, and A. Webster. Querying a building information model for construction-specific spatial information. *Advanced Engineering Informatics*, 26(4):904 – 923, 2012. EG-ICE 2011 + SI: Modern Concurrent Engineering.

[99] FAA. Air traffic control, faa order jo 7110.65w.

[100] M. R. Gary and D. S. Johnson. Computers and intractability: A guide to the theory of np-completeness, 1979.

[101] S. Arora and B. Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[102] Tesla Motors Inc. Tesla superchargers, 2016. Accessed: 11/10/2016.

[103] D. Cattaruzza, N. Absi, and D. Feillet. Vehicle routing problems with multiple trips. *4OR*, 14(3):223–259, 2016.

[104] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097 – 1100, 1997.

[105] P. Hansen and N. Mladenović. Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449 – 467, 2001.

[106] M. Yazdani, M. Amiri, and M. Zandieh. Flexible job-shop scheduling with parallel variable neighborhood search algorithm. *Expert Systems with Applications*, 37(1):678 – 687, 2010.

[107] S. Muelas, A. LaTorre, and J.-M. Peña. A variable neighborhood search algorithm for the optimization of a dial-a-ride problem in a large city. *Expert Systems with Applications*, 40(14):5516 – 5531, 2013.

[108] A. Stenger, D. Vigo, S. Enz, and M. Schwind. An adaptive variable neighborhood search algorithm for a vehicle routing problem arising in small package shipping. *Transportation Science*, 47(1):64–80, 2013.

[109] J. W. Escobar, R. Linfati, M. G. Baldoquin, and P. Toth. A granular variable tabu neighborhood search for the capacitated location-routing problem. *Transportation Research Part B: Methodological*, 67:344 – 356, 2014.

[110] D. Aksen, O. Kaya, F. S. Salman, and ˙ An adaptive large neighborhood search algorithm for a selective and periodic inventory routing problem. *European Journal of Operational Research*, 239(2):413 – 426, 2014.

[111] P. Hansen and N. Mladenović. *An Introduction to Variable Neighborhood Search*, pages 433–458. Springer US, Boston, MA, 1999.

[112] W. J. Gutjahr. *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming*, chapter Convergence Analysis of Metaheuristics, pages 159–187. Springer US, Boston, MA, 2010.

[113] A.W. Johnson and S.H. Jacobson. On the convergence of generalized hill climbing algorithms. *Discrete Applied Mathematics*, 119(1–2):37 – 57, 2002. Special Issue devoted to Foundation of Heuristics in Combinatoria l Optimization.

[114] X.-F. Zhang, M. Koshimura, H. Fujita, and R. Hasegawa. Hybrid particle swarm optimization and convergence analysis for scheduling problems. In *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, pages 307–314. ACM, 2012.

[115] X. Chen and Y. Li. On convergence and parameter selection of an improved particle swarm optimization. *International Journal of Control, Automation, and Systems*, 6(4):559–570, 2008.

[116] X. Song. *Hybrid particle swarm algorithm for job shop scheduling problems*. INTECH Open Access Publisher, 2010.

[117] F. Hillier and G. J Lieberman. Introduction to operations research. *McGraw-Hill New York*, 2010.

[118] S. M. Ross. *Introduction to probability models*. Academic press, 2014.

[119] A. Rezaee Jordehi and J. Jasni. Parameter selection in particle swarm optimisation: a survey. *Journal of Experimental & Theoretical Artificial Intelligence*, 25(4):527–542, 2013.

[120] X. Cai, J. Zeng, Y. Tan, and Z. Cui. *Individual parameter selection strategy for particle swarm optimization*. INTECH Open Access Publisher, 2009.

[121] Y. Shi and R. C Eberhart. Parameter selection in particle swarm optimization. In *International Conference on Evolutionary Programming*, pages 591–600. Springer, 1998.

[122] M. Jiang, Y. P. Luo, and S. Y. Yang. Stochastic convergence analysis and parameter selection of the standard particle swarm optimization algorithm. *Information Processing Letters*, 102(1):8–16, 2007.

[123] M. R. Lohokare, S. S. Pattnaik, S. Devi, B. K. Panigrahi, S. Das, and D. G. Jadhav. *Discrete Variables Function Optimization Using Accelerated Biogeography-Based Optimization*, pages 322–329. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[124] OpenMP. Openmp® library website, 2016. Accessed: 19/12/2016.

[125] ENAV S.p.A. Enav website, 2016. Accessed: 24/10/2016.

[126] Air Traffic Control The Netherlands. Ais netherlands website, 2016. Accessed: 25/10/2016.

[127] S. Niemeijer and G. Valé. Functional design of dynamic taxi-time prediction sub-project of timeline at amsterdam schiphol airport.

[128] H. Khadilkar and H. Balakrishnan. Estimation of aircraft taxi fuel burn using flight data recorder archives. *Transportation Research Part D: Transport and Environment*, 17(7):532 – 537, 2012.

[129] Airport Cooperative Research Program, United States. Federal Aviation Administration, and Environmental Science Associates. *Handbook for Evaluating Emissions and Costs of APUs and Alternative Systems*, volume 64. Transportation Research Board, 2012.

[130] IATA. International air transport association website, 2017. Accessed: 16/01/2017.

[131] Z. Bar-Yehuda. plot google map matlab function, June 2010.